The Train Driver Recovery Problem - Solution Method and Decision Support System Framework

Rezanova, Natalia Jurjevna

Publication date:
2009

Document Version
Publisher's PDF, also known as Version of record

The Train Driver Recovery Problem – Solution Method and Decision Support System Framework

Natalia J. Rezanova

Kgs. Lyngby, Denmark, 2009

DTU Management Engineering
Department of Management Engineering
Technical University of Denmark

Produktionstorvet, building 424
DK-2800 Kgs. Lyngby, Denmark
Phone +45 45 25 48 00, Fax +45 45 25 48 05
www.man.dtu.dk

Preface

This thesis is prepared at DTU Management Engineering Department, the Technical University of Denmark, in partial fulfillment of the requirements for acquiring the Degree of Doctor of Philosophy (Ph.D.) in Engineering Science.

The main focus of this thesis is on state-of-the-art Operations Research methods applied within the area of disruption management in public railway industry, particular within the train driver recovery. An optimization solution method to the train driver recovery problem is proposed, and a prototype for train driver dispatchers decision support system is developed. The project is carried out in a cooperation with a Danish passenger railway operator DSB S-tog A/S.

The Ph.D. project is supervised by professor Jens Clausen, who is also a part-time chief analyst at DSB S-tog A/S. A substantial scientific contribution to this project is provided by the supervision of professor David M. Ryan, Department of Engineering Science, The University of Auckland, New Zealand, who is also a visiting professor at the Technical University of Denmark.

Kgs. Lyngby, Denmark, May 2009

Natalia J. Rezanova

Abstract

In this thesis we consider the *train driver recovery problem* (TDRP). The problem occurs when the daily train driver schedule becomes infeasible due to irregular operations on the railway network. Unforeseen disruptions such as signalling problems or rolling stock failures prevent the train drivers from following the originally scheduled sequence of activities in their duties. The real-time re-scheduling of the disrupted train driver duties is currently performed manually by the train driver dispatchers. If the disruption is severe and many train driver duties are disturbed, this is a very complicated task to carry out. The interest of the passenger railway operator DSB S-tog A/S in introducing automated decision support for the train driver dispatchers is a key motivation for this project.

We propose an optimization-based solution method for solving the TDRP and develop a prototype for the decision support system. The framework is based on solving restricted TDRP instances with a rolling time horizon, aiming at modifying the original duty schedule as little as possible. We formulate TDRP as a set partitioning model, where variables represent train driver recovery duties, and describe why the proposed model and solution method is suitable for solving in real-time. Recovery duties are generated as resource constrained paths in duty networks, and the set partitioning problem is solved with a linear programming based branch-and-price algorithm. Dynamic column generation and problem space expansion at each node of the branch-and-price tree together with a constraint branching strategy con-

tribute to the solution method.

Real-life operational data is provided by DSB S-tog A/S in order to test the implemented solution method. Based on the computational experiments presented in this thesis, we conclude that the proposed approach is indeed applicable for implementation in a decision support system for train driver dispatchers in practice. DSB S-tog A/S is working on using the research results obtained during this thesis and the programming code of the prototype to develop and implement the train driver decision support system in their operational environment.

Besides solving a particular optimization problem, this thesis contributes with a description of the railway planning process, tactical crew scheduling and the real-time dispatching solutions, taking a starting point in DSB S-tog's operations. Furthermore, we present comprehensive reviews of operations research applications within railway crew scheduling, rolling stock re-scheduling, railway crew re-scheduling, and airline crew recovery. In addition, the project has resulted in the three scientific publications listed below.

1. Rezanova NJ, Ryan DM. The train driver recovery problem–A set partitioning based model and solution method. Computers and Operations Research, in press, 2009. doi: 10.1016/j.cor.2009.03.023.

2. Clausen J, Larsen A, Larsen J, Rezanova NJ. Disruption management in the airline industry–Concepts, models and methods. Computers and Operations Research, in press, 2009. doi: 10.1016/j.cor.2009.03.027.

3. Rezanova NJ, Ryan DM. The train driver recovery problem–A set partitioning based model and solution method. IMM-Technical Report-2006-24. Informatics and Mathematical Modelling, Technical University of Denmark, 2006. Available at http://www2.imm.dtu.dk/pubdb/p.php?5157.

Resumé

Hovedformålet med denne afhandling er at anvende state-of-the-art operationsanalytiske metoder inden for realtidsdisponering ifm. jernbanedrift. Arbejdet er udført i samarbejde med DSB S-tog A/S og omhandler problemet med genopretning af tjenesteplaner for lokomotivførere (*the train driver recovery problem*, TDRP). Problemet opstår, når den daglige tjenesteplan for lokomotivførere ikke længere kan opretholdes på grund af uregelmæssigheder på jernbanenetværket. Uventede driftsforstyrrelser såsom signalfejl eller nedbrud af tog forhindrer lokomotivførere i at følge deres oprindelige tjenester. Realtidsdisponering og genopretning af de afbrudte tjenester udføres på nuværende tidspunkt manuelt af personaledisponenter. Omfættende driftsforstyrrelser påvirker mange tjenester, og genopretning af lokomotivførerplanen bliver kompliceret og uoverskuelig. Hovedmotivationen for dette projekt er DSB S-togs interesse for at udvikle et automatisk beslutningsstøttesystem til personaledisponenterne er .

Vi designer en optimeringsbaseret løsningsmetode til TDRP og udvikler en prototype til beslutningsstøttesystemet. Fremgangsmåden er at løse små TDRP instanser over en rullende tidshorisont. Hver TDRP er rettet mod at bygge genopretningstjenester samtidig med at ændre den oprindelige tjenesteplan så lidt som muligt. Vi formulerer TDRP som et set partitioning problem, hvor beslutningsvariablene repræsenterer genopretningstjenester. Vi beskriver, hvorfor den forslåede model og løsningsmetoden er velegnet til realtidsdisponering. Genopretningstjenesterne genereres som korteste

veje med ressourcebegrænsninger i et specielt byggede tjenestenetværk. Set partitioning problemet løses med en branch-and-price metode. Dynamisk søjlegenerering og dynamisk udvidelse af løsningsrummet sammen med constraint branching som forgreningsstrategi bidrager til løsningsmetodens effektivitet.

DSB S-tog A/S har leveret operationelt datagrundlag til at teste den implementerede løsningsmetode. Baseret på de gennemførte eksperimenter, konkluderer vi, at løsnings-metoden i høj grad er egnet til beslutningsstøtte i praksis. DSB S-tog A/S arbejder på at bruge de videnskabsmæssige resultater og programkoden fra den udviklede prototype til udvikling af beslutningsstøtte i det operationelle miljø.

Udover at løse et bestemt optimeringsproblem, giver denne afhandling med udgangs-punkt i DSB S-togs drift en beskrivelse af planlægningsprocesser i jernbanedrift, mandskabsplanlægning på det taktiske niveau og realtids disponeringsstrategier. Ydermere præsenterer vi omfattende redegørelser for operationsanalytiske anvendelser med referencer inden for mandskabsplanlægning ifm. jernbanedrift, materieldisponering, og mandskabsdisponering inden for jernbane og luftfart. Afhandling har desuden resulteret i følgende tre publikationer:

1. Rezanova NJ, Ryan DM. The train driver recovery problem–A set partitioning based model and solution method. Computers and Operations Research, in press, 2009. doi: 10.1016/j.cor.2009.03.023.

2. Clausen J, Larsen A, Larsen J, Rezanova NJ. Disruption management in the airline industry–Concepts, models and methods. Computers and Operations Research, in press, 2009. doi: 10.1016/j.cor.2009.03.027.

3. Rezanova NJ, Ryan DM. The train driver recovery problem–A set partitioning based model and solution method. IMM-Technical Report-2006-24. Informatics and Mathematical Modelling, Technical University of Denmark, 2006. Available at http://www2.imm.dtu.dk/pubdb/p.php?5157.

Acknowledgements

This research project would not have been possible without efforts of several people. I would like to thank my supervisor professor Jens Clausen for giving me an opportunity to write this thesis, for the professional advice and personal support throughout the project. A very special word of gratitude goes to professor David Ryan for providing the fundamental idea to the solution method implemented in this thesis and for supervising my work during and after my stay at The University of Auckland, New Zealand. It has been a great honour and a great pleasure working with you, David.

I would like to thank DSB S-tog A/S for providing part of the financial support for this research project, and particulary Steen Larsen, the former Head of the Production Planning department, for starting up this project and for being a visionary manager. I warmly thank all my former colleagues at the Production Planning at S-tog, and particulary the Analysis Group, for being fantastic colleagues, for answering my questions and contributing to my knowledge about the railway operations on the S-train network in general and the train driver scheduling at S-tog in particular. A special thanks goes to Eva Ryom, a train driver dispatcher at DSB S-tog A/S, for introducing me to the current real-time dispatch practice and for the helpful comments on recovery solutions produced by my program.

I would like to thank developers at Mosek ApS for an outstanding and prompt customer support regarding the linear programming solver with a .NET Ap-

plication Programming Interface, for a comprehensive user manual, and for providing a free-of-charge software license for graduate students.

To all my friends and colleagues at DTU, thank you for creating a wonderful working and social environment, for our fruitful discussions, and for always being ready to help and support. A special acknowledgement to Wen Min and Ma Guilin, with whom I shared many late working hours and many delicious meals prepared by Min, and to Richard Lusby, for proof-reading parts of my thesis and for the valuable comments.

A personal gratitude to my husband Jesper Larsen, for your endless moral support, for always being there for me, for taking care of our children and keeping our family running when I was too busy or too frustrated. This thesis would not have reached its existence without your presence.

Natalia J. Rezanova

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Railway Optimization

This chapter provides a general introduction to the planning process in the passenger railway transportation with references to models and applications within railway optimization. The tactical railway crew planning optimization models and operations research applications are reviewed in order to introduce the reader to the field of train driver scheduling. A general introduction to operations of DSB S-tog A/S is presented. The train driver planning process at DSB S-tog A/S is described in detail, and different components of the train driver schedule are presented.

1.1 Passenger Railway Planning Process

The European railway transportation has been suffering a decline since the 1970's (source: www.ec.europa.eu/transport/rail). Strong competitiveness from the road freight transport has been the main reason for a drastic decline in the freight railway transportation, while the expansion of the low cost airlines has shown an effect on the passenger railway transportation. In order to reverse the decline, the European Commission focuses on supporting targeted railway research projects, while many European railway operators start to invest in extensive computer-aided decision support in all areas of the planning process.

The planning processes in the passenger railway transportation, organized by the time horizon of the planning steps, are illustrated on Figure 1.1. Huisman et al. [2005] also distinguish between central and local impacts of the planning problems on different steps of the planning process.

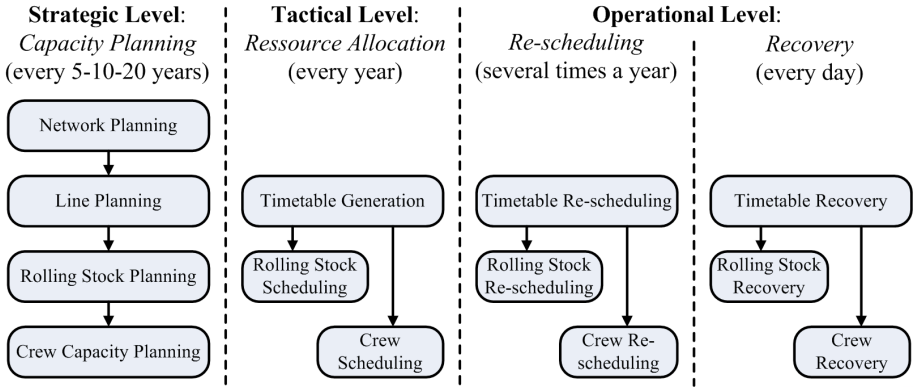


Figure 1.1: Passenger railway transportation planning process.

Operations research methods are applied in all areas of the planning process of the railway transportation industry. Assad [1980] and Assad [1981] present the first surveys of optimization-based models applied within the railway environment. Haghani [1987] gives an overview on optimization methods within the freight railway transportation, while Bussieck et al. [1997], Huisman et al. [2005], Caprara et al. [2007] review operations research methods

within the passenger railway industry.

Strategic Level

Network planning decisions involve expansions and modifications to the existing railway network. These are conducted by the national rail authorities in a cooperation with infrastructure managers. Due to the impact of the infrastructure modifications on the transportation system and the high level of investments, such decisions are only made when considered to be absolutely necessary for improving the national transportation system. Review of the line planning models can be found in e.g. Goossens [2004].

From the train operator point of view, strategic decisions involve planning of the train line pattern in a cooperation with the infrastructure manager, which implies decisions for changing the capacity of the rolling stock and the number of crew. If a train operator estimates that the current timetable does not correspond to the passenger demand or other parameters, a new timetable can be generated. Successful implementation of completely new timetables in Berlin Underground and the Netherlands Railways are reported by Liebchen [2008] and Kroon et al. [2009], respectively. The latter implementation was awarded with the 2008 Franz Edelman Award for Achievement in Operations Research and the Management Sciences. Folkmann et al. [2007] present a capacity assignment model for the rolling stock, which can be used on a strategic level to decide the optimal fleet capacity for a given timetable and a passenger demand according to different objectives, and a manpower planning model, which can be used to estimate the number of drivers necessary for covering a daily timetable.

Tactical Level

The tactical level involves decisions about the assignment of company's resources on a yearly basis. Every year, a "new" timetable is generated. The timetable is only a slight modification from the one of the previous year. Peeters [2003] and Liebchen [2006] provide exhaustive overviews of models, methods and publications within periodic (cyclic) timetabling. When a timetable is presented, the rolling stock schedule and the crew schedules for train drivers and conductors are generated, often by adopting the previous schedules to the new operations. The assignment of the rolling stock

and railway crew on the tactical level is to a certain extend covered in the operations research literature, and many train operators use optimization methods for resource planning. Cordeau et al. [1998] present models for the train routing and scheduling. A review of the latest publications regarding the rolling stock planning can be found in e.g. Maróti [2006] and Jespersen Groth [2008a]. Caprara et al. [1997] focus on the algorithms for the railway crew management. A more contemporary overview of railway crew planning applications on the tactical level is presented in Section 1.2 of this thesis.

Operational Level

Operational level of the planning process can be divided in two parts: re-scheduling and recovery. Re-scheduling involves adjustments to the schedules due to e.g. maintenance work on the railway network. Recovery takes place on the day of operation in real-time as a consequence of disruptions on the network. Jespersen Groth et al. [2007] provides a review and applications within disruption management in the public railway transportation. A thorough review of crew recovery applications within the railway and the airline industries, as well as some references to other disruption management issues of the railways can be found in Chapter 2 of this thesis.

1.2 Railway Crew Scheduling

The railway crew workforce consists of train drivers, conductors and shunting personnel and accounts for one of the most important resources for any railway operator. The railway company has to ensure a certain number of railway crew members present for every train departure at any time during the operation. Railway crew management covers all planning levels: strategic, tactical and operational.

Strategic crew management deals with strategic issues related to the long term capacity planning of train drivers, conductors and shunting personnel. Capacity issues include decisions about hiring or firing crew, training and education of crew to reach flexibility in covered tasks, opening or closing crew depots, re-allocating crew among depots in order to achieve a balanced

crew composition with respect to age, gender and skills. Strategic issues are dealt with continuously and take time to implement.

Railway crew management on a *tactical* planning level is focused on allocation of existing manpower resources, e.g. train drivers and conductors, to train tasks (trips) in the planned train timetable. The railway crew works in shifts. A work shift contains a duty, which is a sequence of train tasks, meal breaks and other activities. Traditionally, the manpower planning is divided into two sequentially solved problems. The *crew scheduling problem* is aimed at finding a minimum cost set of duties which covers one operational period, i.e. one day or one week, dependent on the size of the schedule. All train tasks in the given period must be covered, and the duty schedule must satisfy train driver union rules, such as meal break lengths, duty lengths etc. and operational requirements, such as minimum connection times from one task to another. In the *crew rostering problem* the duties generated and selected during the crew scheduling process are put together into sequences to form rosters. A *roster* is a set of duties with rest periods in between. Many passenger railway operators plan with cyclic rosters. The duty sequence of a *cyclic* roster is performed by n drivers corresponding to the number of weeks in the roster. Each driver performs the same sequence of duties and the sequences are shifted with a one week interval. In other words, during the first calendar week driver 1 performs week 1 of the roster, while driver 2 performs week 2 of the roster, ..., and driver n performs week n of the roster; during the second calendar week driver 1 performs week 2 of the roster, driver 2 performs week 3 of the roster, ..., and driver n performs week 1 of the roster, and so forth. On the calendar week $n+1$ the roster cycle starts again. The crew rostering problem is usually solved separately for each crew depot. One or more rosters are associated with every crew depot. The planning horizon is between six months and one year, and scheduling and rostering is performed for every new timetable. Every roster has to comply with the train driver union contracts and operational regulations of the railway operator, and many people are involved in the tactical crew planning.

The *operational* or *short-term* planning is focused on re-scheduling the originally planned crew schedules as a consequence of e.g. planned track maintenances, special adjustments during public holidays or other events influencing the standard crew schedule. The operational planning is focused on feasibility issues rather than on the minimum cost planning. Crew re-scheduling

can be performed several times during a year and small changes to the daily duty schedule is often performed the day before operation.

The *real-time dispatching* or *recovery* is performed on the day of operations in order to react on unpredicted changes in the timetable as a consequence of disruptions on the train network. The goal of the recovery is to repair those duties that become infeasible because of disruptions. In a recovery situation many scheduling rules are relaxed and the main focus is again on the schedule feasibility, i.e. on ensuring crew availability for as many train departures as possible.

1.2.1 Crew Planning Models

The major part of the operations research applications within railway crew management is focused on the tactical planning issues. Crew scheduling and rostering problems have historically been solved sequentially since they are often too large to be solved simultaneously, and due to the fact that rostering is usually performed separately for each crew depot. The first survey on crew scheduling and rostering models in the railway industry is presented in Caprara et al. [1997] and later extended in a section on crew planning in Caprara et al. [2007]. An annotated bibliography of scheduling within railways can be found in Ernst et al. [2004].

The manpower scheduling in the railway industry is similar to other public transportation industries, i.e. airlines and bus, tram, and metro companies. Operations research methods have been widely applied within the crew management in the airline industry and to some extent in the mass-transit systems. For a comprehensive overview of the models and solution methods of the airline crew scheduling we refer to Barnhart et al. [2003]. Crew management applications in mass-transit are covered in e.g. Wilson [1999].

The Crew Scheduling Problem

The railway crew scheduling problem can be formulated as a set covering problem with side constraints. We here present a generalized problem formulation presented by e.g. Kroon and Fischetti [2000] and Abbink et al. [2005]. Let T represent the train trips to be covered, D be the set of potential driver duties and R represent the additional constraints, which cannot be explicitly covered in the driver duty generation. A binary decision variable x_d equals one if duty $d \in D$ is selected in the optimal solution and equals zero otherwise.

$$\text{Minimise} \quad \sum_{d \in D} c_d x_d \tag{1.1}$$

$$\text{Subject to} \quad \sum_{d \in D} a_d^t x_d \geq n \qquad \forall t \in T, \tag{1.2}$$

$$\sum_{d \in D} b_d^r x_d \leq u_r \qquad \forall r \in R, \tag{1.3}$$

$$x_d \in \{1, 0\} \quad \forall d \in D. \tag{1.4}$$

Each duty has a cost c_d. Duty costs can represent e.g. penalties for passengering tasks or too much idle time in the duty. Violations of different soft constraints according to the train driver labor union rules can also be expressed through the duty cost. The objective function (1.1) minimizes the total cost of the schedule. If costs c_d are equal for all duties in D, the objective function minimizes the number of duties in the schedule.

A binary parameter a_d^t is used to define whether or not the trip $t \in T$ is covered by the duty $d \in D$. The set covering constraints (1.2) force each train trip to be covered by at least n duties, where n is the required number of personnel for covering each trip. If a trip is covered by more than the required of number of personnel in the optimal solution, the excess drivers or conductors are assigned to a passengering task (deadheading) on that train trip. The set of resource constraints (1.3) is used to cover additional requirements not related to individual duties, but to certain forced or forbidden combinations of duties. These requirements are usually expressed on a depot level, like the

maximum average length of the duties per depot or the maximum number of duties per depot. Fair allocation of different workloads can also be expressed through the resource constraints. If u_r is the availability of a certain resource $r \in R$, then the binary parameter b_d^r describes the amount of the resource r used by the duty d.

The Crew Rostering Problem

The railway crew rostering problem can be formulated as a network flow problem with side constraints on a complete directed graph $G = (V, A)$, where the set of vertices V represent train driver duties and arcs in A represent the consecutive sequencing of duty pairs within a roster. A circuit in graph G represents a feasible roster. Let $\delta^+(i)$ and $\delta^-(i)$ represent the set of arcs leaving and entering node $i \in V$, respectively. Let $\mathcal{P}$ be a family of arc sets P, containing arcs in A, which are infeasible to cover for a crew for some operational reason. In order to formulate the problem as a linear integer problem, a binary variable x_{ij} is introduced for each arc $(i, j) \in A$, $x_{ij} = 1$ if the sequence of duties (i, j) is in the solution and $x_{ij} = 0$ otherwise. The following problem formulation is due to Caprara et al. [1997] and Caprara et al. [2007].

$$\text{Minimise} \quad \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{1.5}$$

$$\text{Subject to} \quad \sum_{(i,j)\in\delta^+(i)} x_{ij} - \sum_{(i,j)\in\delta^-(i)} x_{ij} = 0 \qquad \forall i \in V, \tag{1.6}$$

$$\sum_{(i,j)\in\delta^+(i)} x_{ij} = 1 \qquad \forall i \in V, \tag{1.7}$$

$$\sum_{(i,j)\in P} x_{ij} \leq |P| - 1 \qquad \forall P \subset \mathcal{P}, \tag{1.8}$$

$$x_{ij} \in \{1, 0\} \qquad \forall (i, j) \in A. \tag{1.9}$$

The problem finds a feasible set of rosters, covering all the duties and minimising the overall length of the rosters. The first two sets of constraints

ensure that each node in the graph is covered by exactly one circuit. The set of constraints (1.8) forbid the choice of those arc sequences, which cannot be covered by the same crew due to operational constraints. According to Caprara et al. [2007], the network flow formulation is more suitable for the railway crew rostering than the generalized set partitioning formulation used for e.g. airline crew rostering (Ryan [1992], Gamache et al. [1999], Kohl and Karisch [2004], among others) due to the difficulties in using column generation techniques, "probably due to the combination of the relatively large number of duties in a roster and the very complicated operational constraints imposed on a roster".

1.2.2 Railway Crew Scheduling Applications

The purpose of this section is to highlight operations research applications within tactical railway crew scheduling and rostering. The literature within the railway crew planing spans from descriptions of models and solution methods to prototypes and implementations of decision support systems in railway companies. Most publications concern implementations exclusively for one particular railway operator, whereas a few crew scheduling systems were successfully implemented in more than one railway company, as well as within airline and mass-transit industries. On the other hand, some train operators use systems developed by different research groups and software vendors. Therefore, this tactical railway crew planning review is structured by describing publications either for a particular software package or for a particular railway company, or both.

TRACS II

TRACS II, which is an acronym for Techniques for Running Automatic Crew Scheduling, is a driver scheduling system developed at the University of Leeds, Great Britain. TRACS II originated from a bus driver scheduling system IMPACS, which was successfully implemented in large bus companies in UK (see Wren and Kwan [1999] and Meilton [2001]). A pilot project in a collaboration with the Operational Research Unit of British Rail has

shown a big advantage of using TRACS II for the train driver scheduling (see Parker et al. [1995], Wren et al. [1994]). Since then the system has been employed by several bus, train and metro companies. The historical development of the system can be followed in Kwan et al. [1996], Kwan et al. [1999a], Kwan et al. [2004] and Wren [2004].

The overall approach to solving the crew scheduling problem in TRACS II is to generate a large number of potential driver duties using several preprocessing techniques (see, for example, Kwan et al. [1999a] or Smith et al. [2001]), reduce the size of the generated set if necessary and select a subset of duties which covers the train tasks in the timetable according to predefined objectives, like cost minimisation. The solution method applied in the system is described in Fores et al. [1999], Fores et al. [2001], Fores et al. [2002], Wren et al. [2003]. The selection of duties is performed by solving a set covering crew scheduling problem. The optimisation module is originally based on a set partitioning problem solver ZIP developed by Ryan [1980]. The solver was further developed by Smith and Wren [1988] and Wren and Smith [1988] for IMPACS. Recent development and advances of this component are described in details in Fores et al. [1999]. A linear programming relaxation of the crew scheduling set covering model is solved using a column generation strategy developed by Fores [1996]. A branch-and-bound procedure by Smith and Wren [1988] is used to find integer solutions. An alternative dual approach for solving the crew scheduling problem is described in Willers [1995] and Willers et al. [1995]. Kwan et al. [1999b], Kwan et al. [2000], Kwan et al. [2001] describe alternative approaches, where genetic algorithm heuristics are used to reduce the problem size of the crew scheduling problem, and where the information of the linear programming relaxation of the set covering problem is exploited.

The Italian Railway Company

In 1994 and 1995 the Italian Railway Company (Ferrovie dello Stato S.p.A.) promoted two competitions in a cooperation with the Italian Operational Research Society. The first competition was to design and implement an effective algorithm for the set covering problem used to formulate and solve large scale crew scheduling problems. The second competition concerned

the rostering optimization problem. The winners of both competitions came from the Dipartmimento di Elettronica, Informatica e Sistemistica of the University of Bologna. Caprara et al. [1999a] describe the crew planning problem at Italian Railway Company and how it is solved by the methods developed by the winners of the two competitions.

Details of the crew scheduling problem solution method implemented by the winners to the first competition, Caprara et al. [1999b], are presented in the next section. The winner of the second competition, Caprara et al. [1998b] describe in details the model and solution method for the railways crew rostering problem. The crew rostering problem is formulated as a network flow problem with side constraints. A circuit in the network corresponds to a roster. The side constraints forbid combinations of arcs which define infeasible rosters. The problem finds a feasible set of rosters, covering all the duties and minimizing the total number of weeks in the rosters. A relaxation of the integer problem formulation of the crew rostering problem is solved. Constraints which forbid certain rosters are removed and the side constraints which cover additional requirements to the amount of rest periods in a roster are relaxed in a Lagrangian way. A heuristic approach is used to build rosters, using the lower bound computations obtained by solving the Lagrangian dual problem. The rosters are built in a sequential way. The choice of the next duty to be added to a roster is based on a score, where the lower bound value of the solution after adding the duty is taken into account and some other penalties are used. If the roster is feasible after assigning a duty, it becomes a candidate to be chosen as the current best roster. The procedure is iterated until no better roster than the current best can be constructed. When the roster is chosen, all duties included in the roster are removed from the problem. The process is iterated until all duties have been sequenced. A refining procedure can be used to improve the solution. Applied to the crew rostering problem of the Italian Railway Company, the heuristic produced very good results, and the algorithm was implemented in the decision support system of the company.

A combination of constraint logic programming (CLP) and operations research methods applied to the crew rostering problem of the Italian Railway Company is reported in Caprara et al. [1998a]. CLP is a combination of logic programming and constraint solving. Recent advances in CLP can be found in e.g. Azevedo et al. [2007]. The authors use CLP to obtain a heuristic

solution for the crew rostering problem. The lower bound of the problem, expressed as the number of weeks in the roster, is found with the OR solution method proposed in Caprara et al. [1998b]. Several strategies are used in order to limit the search space. The obtained solution is improved in a post-optimization procedure by applying local changes to the feasible solution. The authors stress that compared to the pure OR method, the proposed solution procedure is much easier to model and program. Experiments show that the solution quality and the running times of the pure OR and the CLP/OR methods are comparable. The post-optimization procedure significantly improves the initial solution, but requires about half of the computational time and therefore needs to be refined.

Caprara et al. [2001] point out that it has been evident through the previous research that a feedback between the crew scheduling phase and the crew rostering phase may significantly improve the quality of the final solution. The authors present a new crew pairing generator developed within the EU Project TRIO, as well as techniques for integrating the scheduling and rostering optimization. In the integrated approach the duty selection in the crew scheduling phase is driven by the objective function of the crew rostering problem, i.e. every time a new candidate set of subsequent duties is found, it is checked whether this candidate leads to a set of rosters better than the incumbent one.

TURNI and Crew Scheduling at the Netherlands Railways

An automatic crew scheduling software TURNI has been developed by an Italian company Double-Click s.a.s. The optimization module in TURNI is based on an effective heuristic method for the set covering problem by Caprara et al. [1999b], developed for the crew scheduling competition of the Italian Railway Company described above. The heuristic is based on the Lagrangian relaxation, where the Lagrangian multipliers are found with a subgradient optimization and a heuristic procedure is applied to generate several near-optimal multiplier vectors, which are used to find the best set covering incumbent solution. The incumbent is further improved by column fixing. The implementation of TURNI at DSB S-tog A/S is described in Section 1.4.3.

TURNI has also been successfully implemented at NS Reizigers, the passenger division of the Netherlands Railways. Kroon and Fischetti [2000] and Kroon and Fischetti [2001] introduce the first implementation of TURNI, describe the set covering formulation of the crew scheduling problem and briefly account for the solution method. The scheduling module of TURNI consists of a duty generation part and a duty selection (optimization) part. Since the number of potential feasible duties is very large, they are not generated a priori, but "on the fly" during the optimization, using dynamic column generation based on Lagrangian dual information. Many company-related constraints and requirements can be included and easily changed in the duty generation process through a simple user interface. TURNI has been used at Netherlands Railways since 2000 (see Fischetti and Kroon [2001]) and has been contributing to estimated savings of $4.8 million per year according to Abbink et al. [2005] due to the improved crew scheduling. The usage of TURNI has been further developed by the operations researchers at Netherlands Railways by applying iterative partitioning of the large crew scheduling problem. Partitioning methods and results are presented in Abbink et al. [2007] and Abbink et al. [2008]. A success of the crew scheduling improvement has contributed to a successful implementation of the new timetable at Netherlands Railways, the 2008 Franz Edelman Award winner project described in Kroon et al. [2009].

Harmony CDR and Crew Rostering at the Netherlands Railways

A decision support system Harmony CDR (Crew Duty Rostering) for crew planning is developed by ORTEC Consultants, the Netherlands. The system is originally designed to create and maintain rosters for airline crew, and later developed for the railway crew scheduling and rostering. The problems are formulated as generalized set partitioning problems, which are solved with LP-relaxation and branch-and-price. Freling et al. [2004] report insights into the implementation of the branch-and-price algorithm, including efficient data structures, column management and different heuristic strategies for speeding up the computation times. Practical issues of implementing the system in the Netherlands Railways and in a European airline are reported as well. The general framework of the system allows incorporating duty feasibility constraints without interfering with the structure of the

algorithm. A special version of the branch-and-price algorithm applied to the crew scheduling of guards in the Netherlands Railways is presented in Freling et al. [2001]. The authors focus on the duty generation procedure and acceleration techniques of the algorithm.

The crew rostering solution for the catering crews of the High Speed Train is presented in Lentink et al. [2002]. For this particular application a different algorithm than the branch-and-price solution to the set partitioning problem was developed in Harmony CDR. The algorithm is based on successively solving linear assignment problems of assigning duties to crew, and the solution process resembles the manual planning process of the company.

The cyclic crew rostering problem of the Netherlands Railways is described in Hartog et al. [2008]. The authors present the complicated set of labor rules, which have to be taken into account when generating rosters for each crew member. The problem is solved in two stages. First, roster patterns are designed, assigning an early, late or night duty, a rest period or a reserve duty to each day in the roster. Second, specific duties are assigned to fit into the roster patterns. The experiments for rostering one of the largest crew base in the Netherlands are presented. Results are very satisfactory with respect to both the solution quality and computational times.

Hong Kong Light Rail

Chu and Chan [1998] report a network based heuristic method to solve the crew scheduling problem in a decision support system of Hong Kong Light Rail Transit, a passenger railway operator, which is a part of Kowloon-Canton Railway Cooperation. Train tasks are first combined into pieces of work, which can be performed by a driver without rest. Feasible pieces of work are found with shortest path calculations in a network, where relief times/places are represented by nodes and edges represent train tasks. A matching problem is solved to form driver duties, which contain one or two pieces of work. A similar method is reported by e.g. Ball et al. [1983]. Several local search heuristics are used to improve the solution. Only a slight improvement in productivity rate of the train drivers is achieved compared to the manual solution, but the decision support system can save a lot of work and effort

of the schedule planners.

Australian National Rail

A simulated annealing heuristic method for solving the train crew rostering problem of National Rail, the Australian freight train system, is presented by Ernst et al. [1998]. The sparseness of the freight rail network in Australia allows to enumerate the full set of all feasible crew roundtrips (duties). Given the weekly schedule that repeats itself, cyclic rosters can be developed directly from train trips. A set partitioning problem formulation of the crew rostering problem is presented in Ernst et al. [1999]. The large-scale integer program is solved with column generation and cutting planes.

Ernst et al. [2001a] present problem formulations and solution methods to the crew scheduling and the crew rostering problems. The crew scheduling problem formulation includes more sophisticated constraints than the traditional crew scheduling model described in Section 1.2.1, since the sparseness of the train network allows to solve a larger integer programming problem. Solution approaches to finding cyclic and non-cyclic rosters are presented. A similar problem formulation is used by Ernst et al. [2001b] for dealing with a strategic integrated crew scheduling and rostering problem, where the total number of crew and their distribution on the train network is optimized.

Carmen Systems

Resource Management Solution - Rail Crew, abbreviated RMS-R Crew, is an optimization software for railway crew scheduling and rostering developed by Carmen Systems (now owned by Jeppesen, a Boeing subsidiary). The system is implemented in Deutsche Bahn, the German state railways, in the Swedish State Railways and in Green Cargo, a Swedish-based cargo railway operator. Some implementation results at Deutsche Bahn are reported by Kohl [2003]. The rail crew scheduling (pairing) problem is solved as a set partitioning version of the set covering formulation of the problem, with additional hard and soft constraints (see Bengtsson et al. [2004] for the detailed model for-

mulation). The problem is solved with a dynamic column generation, where both LP-relaxation and Lagrangian relaxation approaches are used to obtain dual values. Several techniques are used to generate columns, including the k-shortest paths routine and label merging when generating constrained paths in the pricing network. Integer solutions are provided by a dual-ascent heuristic devised by Wedelin [1995]. The integer strategy is further improved by fixing connections between some tasks and early branching. For details please refer to Bengtsson et al. [2004] and Hjorring [2004].

CREWS

The crew scheduling decision support system CREWS is developed by a Portuguese company SISCOG (Morgado and Martins [1998b]). Among other railway companies, CREWS is installed in the Portuguese Railways under the name ESCALAS (Morgado and Martins [1992] and Morgado and Martins [1993]), in the Netherlands Railways under the name CREWS_NS (Morgado and Martins [1998a]), in the Norwegian State Railways (Martins et al. [2003]) and at S-tog under the name PDS. The system can schedule train drivers and other crew members, e.g. guards. The system has a user-friendly interface and can be used in manual, semi-automatic and automatic modes.

According to Morgado and Martins [1998a], the automatic mode of the system is based on the artificial intelligence method. A modified version of A* algorithm with heuristics to limit the search space is used to build a crew schedule by successively inserting one trip after another. At each stage, candidate tasks are chosen to be added to the part of the schedule generated during previous stages. The choices are guided by an evaluation function, which aims at minimizing the total cost of the schedule. The evaluation function is user-modifiable. Kroon and Fischetti [2000] describe implementation of CREWS in the Netherlands Railways. It is pointed out that even with heuristic methods for limiting the search space, computational times and the required amount of memory are very large, even for moderately sized instances. Due to the fact that CREWS produced scheduling solutions, which were not always satisfactory for the planners, the Netherlands Railways and S-tog have supplemented CREWS with the aforementioned automatic crew scheduling system TURNI.

Taiwan Railway Administration

Lee and Chen [2003] and Lee [2004] present two approaches for solving the train driver scheduling and rostering problem for one crew depot of Taiwan Railway Administration. The first solution approach uses a set covering problem to solve the crew scheduling problem. The problem is solved with an a priori heuristically generated set of duties. If a feasible solution is not achieved, more duties are generated with modified duty generation rules. When a satisfactory duty schedule is built, the rostering problem is solved as a constrained asymmetric travelling salesman problem, minimizing the roster lengths. An IP formulation of the problem without subtour elimination constraints is solved using a commercial optimization software package LINDO (www.lindo.com). Subtours are dealt with heuristically afterwards. The generated duty schedule was more efficient than the manually built one, but the rostering solution was not satisfactory for practical purposes. The second approach is based on solving the scheduling and the rostering problem both sequentially and as an integrated model using a genetic algorithm, which provides a general framework to deal with hard and soft scheduling and rostering constraints. A gene in the algorithm represents a roster, while each number in a gene represents the choice and the sequence of a duty in the roster. Different selection and crossover rules are used in the algorithm. Real-life data is used to fit the parameters of the genetic algorithm. This approach was suitable for solving the problem and gave fair results to the integrated scheduling and rostering problem.

North American Railroad

Vaidyanathan et al. [2007] present a multicommodity network flow approach to the crew scheduling problem for North American railroads. North American railroad networks are much more sparse compared to the European ones, and the crew stays on the same train for much longer time. Another difference is that there is no fixed crew schedule. The crew receives their assignment a few hours before the train departs. The authors ague that column generation approaches to solving the set partitioning/set covering formulations of crew pairing and rostering are not applicable to North American railroads for two

reasons. First, the column generation approach is needlessly complex, since most of the pairings would consist of at most two trains. Second, the solution must be fast enough to be used in a real-time environment due to the operational approach to assigning crew, while the duty rules are very complex, and the subproblem in the column generation scheme would not be as easy to solve as for the European crew scheduling.

In the suggested approach sets of crews governed by the same rules represent commodities, and the flow in a space-time network of individual crew members represent their assignments. The problem is formulated as an integer multicommodity flow model with extra constraints, which ensure the so-called FIFO requirement to crew assignments imposed by the Federal Railway Administration governing North American railroads. The FIFO (first-in-first-out) rule requires that crews should be called on duty in the order they become qualified (ready) for the next assignment at a location. The problem with relaxed FIFO constraints is solved using the MIP solver of ILOG CPLEX 9.0 (www.ilog.com). Two solution approaches for handling FIFO constraints are introduced: a successive constraint generation algorithm (SCG), which iteratively prunes crew assignments that violate the FIFO constraints and a quadratic cost perturbation algorithm (QCP), which penalizes the FIFO violation in a solution. The algorithms are compared with the exact integer programming solution method applied to the original problem formulation. The authors conclude that the QCP algorithm outperforms the other two methods.

London Underground

Sodhi and Norris [2004] present a solution approach to the rostering problem at London Underground. The rostering problem is decomposed into two sequentially solved problems. First, a rest-day pattern of the roster is created, where each day in a roster is represented by either early, late or night duty or a rest day. Second, specific duties are assigned to the pattern obtained in the first stage. The objective of the rest-day pattern generation is to maximize the weighted sum of different types of consecutive days-off, while satisfying union regulations and operational constraints. The rest-day pattern is modelled on a directed graph, where each node represents a weekly pattern of

the roster and the arcs represent legal transitions between patterns. A linear integer problem finds the number of times each pattern (node) should be included in the optimal rest-day pattern by maximizing the preference weights on arcs and nodes. The problem is solved in the commercial solver ILOG CPLEX (www.ilog.com). A final roster pattern is created manually using a depth-first search of the solution to the linear integer problem. The duty assignment to the roster pattern is solved as an assignment problem with side constraints.

1.2.3 Concluding Remarks

The railway industry is not yet such a widely used field of operations research applications as the airline industry, where OR methods have been applied to the crew scheduling problems for several decades and lead to a significant number of scientific publications (Barnhart et al. [2003]). Some railway crew scheduling software packages evolved directly from the crew scheduling solution methods originally developed for the airlines (e.g. Harmony CDR, CREWS or Carmen scheduling solutions) or for bus companies (TRACS II), while others were specifically developed for a particular railway operator (e.g. for the Italian Railway Company or the Australian National Rail). Some crew scheduling applications in the railway industry are more successful than others with respect to implementation in different railway companies, and the solution approaches vary from simple heuristics to more complicated exact and hybrid methods. However, they all contribute to operational cost savings of railway operators due to better crew scheduling solutions or due to the fact that decision support systems improve the daily work of the planners. Moreover, from the operations research point of view, novel approaches to the large-scale optimization problems have been developed by researchers working on solution methods for solving railway crew planning problems. The need for improving services by the railway operators has increased since the railway infrastructure management has been separated from the operation in order to promote an open market and a free competition on the European railway transportation market. Therefore, the demand for operations research applications in the railway industry is far from being fully satisfied, and there is rum for more research and development.

1.3 DSB S-tog A/S and the S-train Network

The Danish railway network shown on Figure 1.2 is operated by the governmentally owned infrastructure manager Banedanmark. Banedanmark was established in 1997 as a consequence of separating the Danish State Railways DSB into two independent companies, the infrastructure manager under the Danish Ministry of Transport and the passenger train operator DSB. The establishment of an independent infrastructure manager took place in order to promote free competition on the Danish railway network. Banedanmark maintains and renews the infrastructure of the network, i.e. tracks, signals, stations, security systems etc., and is responsible for the traffic surveillance and the assignment of the network capacity to the train operators. At the present time there are 16 train operators on the Danish railway network, and 10 of them operate within the passenger traffic (source: www.banedanmark.dk, March 2009).

Figure 1.2: Operators on the Danish railway network.

DSB S-tog A/S (hereinafter referred to as S-tog) is a subsidiary of DSB. S-tog operates on the passenger railway network, which covers the Greater Copenhagen area. Its geographical position is accentuated on the map on Figure 1.2 (source: www.banedanmark.dk) and illustrated as a separate network in the upper right corner of the figure. The network is referred to as the *S-train network* and the trains operated by S-tog are referred to as *S-trains* (see Figure 1.3). The first S-train line opened in 1934. Since then, the infrastructure has expanded into 432 km tracks, where the operational part is covered by 171 km double-tracks almost everywhere except for a 500 meter segment between Værløse and Farum stations. The latter is covered by a single track due to the geographical condition of the segment. The S-train network contains 84 passenger stations. S-tog is the only operator on the S-train network. The train operator is responsible for planning and implementing timetables for the S-trains, maintenance of the rolling stock and crew planning. The operations on the S-train network are very important for the public transportation in the Greater Copenhagen area. S-trains connect to busses, metro, several small train networks, regional and international trains. S-tog transports approximately 300.000 people on a daily basis, counting the 2/3 of all DSB passengers (source: www.dsb.dk, September 2008).

Figure 1.3: S-train at Flintholm station. Photo: Lasse Mølholm.

A schematic view of the S-train network is shown in Figure 1.4. The central part of the network from Dybbølsbro in the southern part of Copenhagen to Svanemøllen in the northern part of Copenhagen is called "the pipe" and includes the Copenhagen Central Station, København H. Six *fingers* of

the network stretch from the central part to Klampenborg, Hillerød, Farum, Frederikssund, Høje Taastrup and Køge. A *circular rail* between Ny Elleberg and Hellerup is the only segment, which does not cross the central part of the network. The schematic view presented in Figure 1.4 does not follow the geographical positions of the stations on the S-train network, but rather emphasises the two *directions* which are used by planners and dispatchers to describe train movements on the S-train network. By convention, direction "north" is represented by Klampenborg, Hillerød, Farum and Hellerup (as a terminal stations of the circular rail), while direction "south" is represented by Frederikssund, Høje Taastrup, Køge and Ny Elleberg stations.

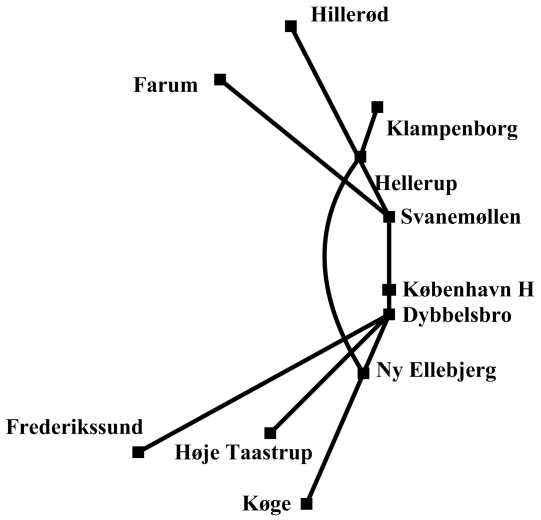


Figure 1.4: Schematic view of the S-train network.

Each segment of the S-train network is covered by at least one *train line*. A line is indicated by a colour and a capital letter, either alone or with a "+" or an "**x**" after the letter. Each train of a line runs back and forth from north to south between two terminal stations. All trains except the trains of the circular rail pass the central segment of the network. Until the end of September 2007 the S-train network was covered by 11 train lines as shown in Figure 2.1(a) on page 24. Throughout this thesis this train timetable is referred to as the *timetable-2007*. Each train line of the timetable-2007 had a cyclic schedule of 20 minutes, i.e. a frequency of 3 trains per hour in each direction. A line was either a main line, running from approximately 5 am

to 1 am next morning, or an extra line indicated by a "+" operating during the daytime hours, from about 6 am to 7 pm, or an extra line indicated by an "**x**", operating during morning and afternoon peak hours. A combination of main lines and extra lines at each segment of the network allowed a higher departure frequency than 20 minutes at almost every station of the network during the daytime hours.

In September 2007 the line pattern of the S-train network was changed to the one shown in Figure 2.1(b). We refer to the new schedule as the *timetable-2008*. Besides from the change in the train line pattern, the number of train lines was reduced to 7, and all "+"-lines were removed. On the other hand, almost every train line of the new schedule has now a cyclic schedule of 10 minutes, i.e. a frequency of 6 trains per hour in each direction during the daytime hours. The new schedule is more stable and is built to enhance passenger satisfaction, since the travel time for passengers travelling on longer distances have decreased, while passengers travelling on short distances can travel with a higher frequency. In the central segment of the network there is a train in each direction every 2nd minute during peak hours.

For the scheduling purposes the planners at S-tog partition the timetable into *trains*, each train having a unique 5-digits number. The first three digits in the train number represent the train line, the stopping pattern (main or extra line) and the direction of the train ("north" or "south"), while the last two digits represent the arrival time interval at København H. A more detailed explanation of S-tog's train numbers can be found in e.g. Jespersen Groth et al. [2006]. A train gets a new number when it starts at a terminal station, e.g. Hillerød station. The number is kept while passing København H until the train reaches the opposite terminal station, e.g. Køge station. When the train turns back, a new number is assigned to it. On weekdays there are 1,303 trains on the network, while there are 963 and 763 trains on Saturdays and Sundays, respectively.

(a) Line pattern of timetable-2007. (b) Line pattern of timetable-2008.

Figure 1.5: The S-train network.

1.4 Tactical Train Driver Planning at S-tog

1.4.1 Train Driver Schedule

At the present time (year 2008) there are approximately 530 train drivers employed by S-tog. A daily S-tog timetable is covered by 256 train drivers on weekdays, 192 train drivers on Saturdays and 160 train drivers on Sundays, excluding scheduled reserve drivers. The remaining part of the workforce is counted for being on vacations, leaves, assigned to rest periods between duties and being on education. Each train driver is able to operate all types of rolling stock owned by S-tog and only one driver is required to operate a train.

Train Driver Duty

A train driver's *duty* is a sequence of tasks (activities), i.e. train drives, meal breaks, riding as a passenger on a train, taxi transfers or stand-by time at the crew depot. The length of a daily duty is usually between 6 and 8 hours with a maximum length of 9 hours. When the duties are generated, the planners aim to build duties of approximately 7 hours, except for weekend duties which can be slightly longer for the sake of free days allocation. A night duty length is at most 8 hours. A night duty is a duty containing working time between 1:30 and 4:30 in the morning. The earliest duty starts at 3:30 in the morning and the latest duty finishes at 2:30 in the following morning. Morning duties are very popular among drivers. There are also noon duties, afternoon duties and evening duties.

A duty starts with a 15 minutes *check-in* and ends with a 10 minutes *check-out* activity at one of the depots. A duty must start and finish at the same *crew depot*, which is a station with rest facilities. Check-in and check-out activities can take place at the four out of five crew depots on the S-train network. The major depot is located at København H and the three smaller check-in depots are situated at Hillerød station in the north of the S-train network, Køge station in the south and at Høje Taastrup station, which was

opened recently.

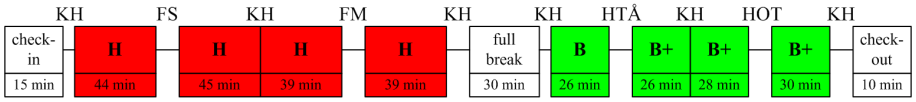


Figure 1.6: A two-blocks duty with merging lines in timetable-2007.

A duty contains either one *long break*, as in the duty shown in Figure 1.6 or two *short breaks* between train tasks, like in the duty shown in Figure 1.8. The length of a long break is 30 minutes. If the break is split into two short breaks, each break must last at least 20 minutes and the total length of the two short breaks in a duty must sum up to 45 minutes. This means that it is feasible to have 20 and 25 minutes or 21 and 24 minutes or 22 and 23 minutes short breaks in the duty. Even though breaks of a longer duration are allowed in practice, any time in excess of the required break length is considered to be a buffer time in the driver duty. A break is held at one of the two crew depots with rest and meal facilities. The largest one is situated at København H and the smallest one is placed at Hellerup station, where drivers assigned to the circular rail segment are scheduled to have their meal breaks. A driver is entitled to a break after at most 3 hours of train driving, with the 3 hours and 30 minutes exception for drivers assigned to lines **H** and **H+** of the timetable-2007, where a return drive between terminal stations Farum and Frederikssund has a total duration of 3 hours and 20 minutes.

A *train task* in a train driver schedule is a train drive between a terminal station and København H or between the two terminal stations of the circular rail. Please notice that the definition of a train task in this project is different from the standard definition of a *train* at S-tog, described in Section 1.3. The same train can be divided into two parts, one from a terminal station to København H and another part from København H to the opposite terminal station of the line, and assigned to two different duties. It is a customary at S-tog to plan the train tasks in a duty in *blocks* with meal breaks in between. There are either two or three blocks in a duty. A block contains one or more *trips*, which are sets of train tasks on one line. A *full trip* is a return train drive from a terminal station on the same train line. For example, a full trip of line **H** in the timetable-2007 is a sequence of four train tasks København H–Frederikssund, Frederikssund–København H, København H–

Farum, Farum–København H. This trip composes the first block of the duty shown in Figure 1.6. A *half trip* is a non-complete trip, and can contain either one, two or three train tasks. As an example, the last block of the duty shown in Figure 1.9 contains only one train task København H–Køge of line **E**. A duty block can contain two "merging" train lines. Two train lines merge, if the train number changes to the number of another line at a terminal station, instead of changing to the subsequent train number of the same line. As an example, lines **B** and **B+** are merging, as illustrated in the second block of the duty in Figure 1.6.

Due to the block structure of duties in the general train driver schedule, the drivers very seldom hand over the train to other drivers at any other stations than crew depots, where the arriving driver either goes on a break, to a reserve task or to a check-out after finishing a trip, while the driver taking over the train comes from a break, a reserve or a check-in activity. This is done to avoid situations, where a train driver is not able to take over another train due to a delay of his/her previous train task, thereby to increase the robustness of the driver schedule. However, during disruptions a driver might be assigned to change to a train task on another train unit at any terminal station or at a rolling stock depot along the train line or even at one of intermediate stations if required from a recovery point of view.

An essential rule for sequencing train tasks in a duty requires that a subsequent train task starts at the station of arrival of the previous train task, and the time between arrival/departure is long enough to allow for a safe connection according to the company regulations and union agreements. For example, a driver change from a train to a meal break at København H must be have a minimum connection time of 5 minutes: 1 minute for handing over the train to another driver and 4 minutes for walking to a depot from the platform. Components of technical connection times with corresponding abbreviation codes are listed in Table 1.1.

It is allowed to schedule *deadheading tasks* in the duties, either as a *passengering task* on timetabled trains or in a *taxi*. Deadheading tasks are usually used for positioning drivers at non-depot terminal stations at the beginning and at the end of the day in order to cover earliest train departures and latest train arrivals or whenever it is otherwise necessary for the schedule feasibility. As an example, the driver duty shown in Figure 1.7 contains a

Table 1.1: Components of technical connection times at S-tog, 2007.

Abbr.	Task Description	Station	Minutes
BEV	Walking between a depot facility and a platform.	Køge	8 min
		Frederikssund	5 min
		Other stations	4 min
KLG	Getting the train ready for the first drive of the day.		5 min
PIF	Handing over a train to another driver.		1 min
PIT	Taking over a train from another driver.		1 min
SPD	Walking to the opposite end of the train at terminal stations.		4 min

taxi deadheading from København H crew depot to Klampenborg station in order to position the driver for the circular rail duty in the beginning of the duty. The duty also contains a passengering task on a train of line **B** from Hellerup to the crew depot at the end of the duty.

The regular duties are constructed such that each driver has a high *variety* in tasks during the day in order to distribute the short and the long train tasks between drivers. As a general rule, a driver is not allowed to drive back and forth between two terminal stations during the whole duty period. Rather, train tasks of different lines must be assigned to each driver during the day, except for duties covering lines **F** and **F+** of the circular rail due to the isolated location of the segment. An example of an unattractive duty containing many train tasks on the circular rail is shown in Figure 1.7. Unattractive circular rail duties are distributed among rosters.

There are different variety levels for different duty types and for different duty schedules. For example, when scheduling general weekday duties allocated to the København H depot, trips on the same train line must not be scheduled in more than one duty block. An example of such a duty is shown in Figure 1.8, where all three duty blocks contain trips on different train lines.

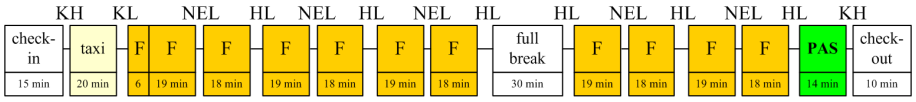


Figure 1.7: A circular rail duty in timetable-2007.

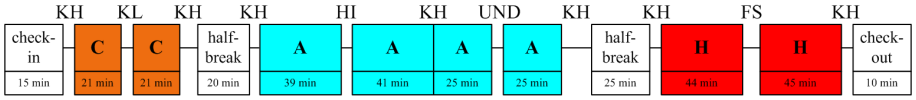


Figure 1.8: A København H depot duty in timetable-2007.

In duties allocated to other depots it is not allowed to schedule trips of the same train line in two consecutive blocks. This rule often forces to schedule Hillerød, Køge and Høje Taastrup depot duties in three blocks with two short breaks in between, since a driver must often use the same train line in order to get back to the depot for the check-out, and driving the same line in two consecutive blocks is not allowed. Figure 1.9 shows an example of a feasible duty belonging to Køge depot.

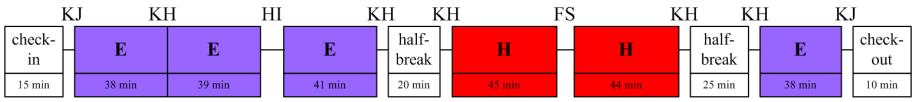


Figure 1.9: A Køge depot duty in timetable-2007.

In special plans for the days with track maintenance the variety rules are relaxed for covering train lines on network segments directly influenced by the maintenance work. Variation in duty tasks implies that many drivers are assigned to each line during the day. As an example, 88 train drivers are assigned to a main train line **E** during a weekday of the planned timetable-2007 and 55 drivers are assigned to the extra line **A+**. A high variations in duties makes it impossible to partition the train driver schedule according to the network segments. This implies that if due to the track maintenance a specific segment of the network is closed, the train driver re-scheduling is not an easy task to perform, since many train driver duties are involved.

Train Driver Roster

S-tog plans with cyclic rosters. A roster at S-tog spans for an even number weeks, since the day off rules are defined for every second week of a roster according to the agreement with the train driver union. There are currently 25 rosters at S-tog. A roster contains a set of duties characterised by some common parameters. For example, a roster can contain all duties with check-in and check-out at the same depot or all early morning duties. There is one roster for each check-in depot (Hillerød, Køge and Høje Taastrup) and 22 rosters assigned to the main crew depot at København H. The smallest roster is covered by eight train drivers (the roster containing early morning duties), the largest rosters are covered by 42 train drivers (e.g. a Hillerød depot roster).

A roster must comply with many requirements. A driver can work for at most 75 hours during every 14 days from Monday to Sunday. A train driver must not work more than seven consecutive days without a rest period. There must be at least one day off every second week of a roster and at least three days off every other second week of a roster including a mandatory free weekend. Hence, a driver always has a free weekend every second week. The days off can for example be scheduled on a Wednesday, a Saturday and a Sunday in one week and on a Thursday next week. There is however a so-called "3–3" roster, where 3 workdays alternate with 3 days off, resulting in longer duties and more days off. Each train driver is entitled to at least 50% evening rest time, i.e. the time between 17:00 in the afternoon and 7:00 in the morning. The evening rest time is calculated as an average for a roster and for a calendar month for reserve drivers. Average working time in a roster must also be satisfied. As a goal, the train drivers have to work 25 hours on average during a roster week. For example, 42 train drivers assigned to the Hillerød depot roster produce 1.050 working hours, which gives 25 hours per roster week on average.

1.4.2 Yearly Planning Process

The yearly train driver planning at S-tog contains five major steps shown in Figure 1.10.

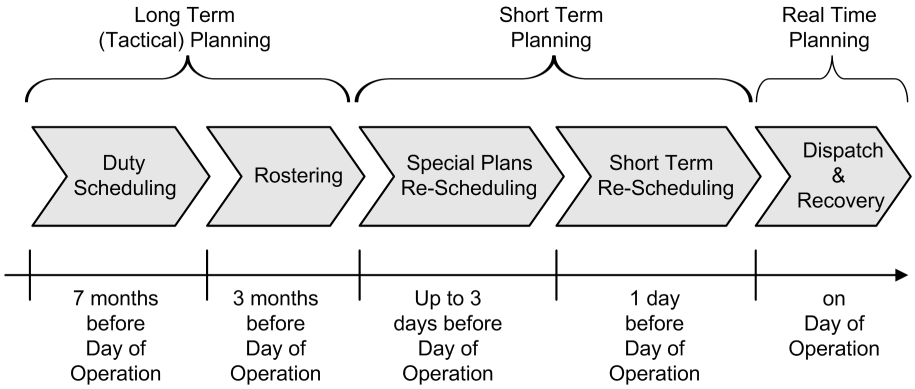


Figure 1.10: Train driver schedule planning steps at S-tog.

At the end of May a new S-tog train timetable valid from January the following year is finished and approved. Every train departure in the timetable must be covered by a driver duty. The *scheduling* process for the train driver schedule starts at least seven months before the launch of the new timetable. During the first month of the scheduling process a scheduling committee discusses if there are any duty rules that need to be adjusted. An example of such an adjustment could be extending the scheduled walking time from a depot to a platform at one of the stations from 2 to 3 minutes. Agreements to adjustments to train driver duty rules are usually reached at the end of June.

During the following month the duty schedule is generated using the automatic scheduling system TURNI (see Section 1.4.3). Three *general* train driver schedules are developed (*normal plan* in Danish): for weekdays, for Saturdays and for Sundays. Deviations from the planned timetable, for instance due to line maintenance or holidays are not taken into account in the general train driver schedules. At the beginning of August the new train driver schedule is sent for approval to the Train Driver Personnel Regional Group for S-tog (*Lokomotivpersonalets Områdegruppe (LPO) S-tog* in Dan-

ish) of the Danish Railway Union (*Dansk Jernbaneforbund* in Danish), which represents train drivers of the S-train network. During the next two-three months the union representatives check the proposed schedule, negotiating closely with the schedulers at S-tog until reaching the final approval of the duty schedule.

When the general duty schedule is approved, the *rostering* process begins. In the beginning/middle of October the primary assignment of duties to rosters takes place. Six train driver representatives gather together "around the table" for three days to negotiate the distribution of all scheduled duties into six pools, where each pool contains several rosters. At the end of October, 25 train driver representatives (one for each roster) gather together for a two-days seminar in order to make the final assignment of duties to rosters. Representatives of each roster pool try to reach feasibility within each particular pool by exchanging duties between rosters in the pool, if it is necessary. If an overall feasibility cannot be reached, some duties might be exchanged between the pools of rosters. The next 8–10 days are used for the finishing touches of the rosters and the anonymous roster set is entered into the scheduling system PDS (see Section 1.4.3), where all scheduling and rostering rules are checked. In the middle of November the new train driver schedule is published and official roster plans are sent to the train drivers.

Over the next three weeks every train driver makes a priority list of all 25 rosters and submit the list to the union representatives. The assignment of drivers to rosters is based on strict seniority. The drivers are listed in decreasing order of length of service at S-tog, and every train driver is assigned to the roster of his/her highest available priority. The staff allocation to rosters is registered in PDS, and in the middle of December the final assignment of train drivers to rosters is finished. The new train driver schedule is launched during the first weekend of January in the following calender year.

Apart from the general schedules, several *special plans* (*sær planer* in Danish) are often necessary in order to take care of the known deviations in the timetable. For example, a special plan is required for the Christmas Eve or for the periods with planned track maintenances on the S-train network. Special plans are not built from scratch. Instead, the general schedules are re-scheduled, taking care of deviations in the timetable caused by planned deviations. In a special plan the start and end time of a duty can only be

moved up to 10 minutes each and at most 10 minutes in total. The content of the duties in the special plan can be changed without notifying the driver up to 72 hours before the start of the day of operations.

On the day before operations the *short term re-scheduling* takes place. Last-minute changes, like sudden sickness of crew and small adjustments in the duties are registered in the system. The short term re-scheduling can be done up to 12 hours before the start of operations. On the day of operation it is often necessary to make adjustments to the daily train driver schedule due to unpredicted disruptions. The *real-time* re-scheduling at the S-train network is described in details in Section 2.1.

1.4.3 Use of Computer-Aided Systems

In September 2001 S-tog and the Portuguese company SISCOG signed a 3-years contract for implementing a crew scheduling system PDS (Personale Disponerings System in Danish), which is a customized version of SISCOG's scheduling system CREWS (Morgado and Martins [1993]). The system should contain four main modules: a long-term scheduling module, a long-term rostering module, a short-term scheduling module (for re-scheduling purposes) and a real-time dispatching module. Manual, semi-automatic and fully automatic functions are to be present in the system.

The system is not yet fully installed, even though the main part of the contract is delivered. The long term planning modules are only used in manual mode, despite the availability of the semi-automatic and the fully automatic functions. The short term scheduling module is only manual. The delivery of the manual mode of the real time dispatching module is due at the end of the year 2008. According to the project plan, the semi-automatic and the automatic modules of the real time dispatch will be delivered at the end of year 2009.

At present PDS at S-tog is used as a data management system. There is a direct information flow between PDS and the personnel data storage system at S-tog. Duty and roster rules are controlled in the Long-Term Data Manager module of PDS as a part of the long term planning. The Staff Allocator com-

ponent of PDS is used to manually update information about which roster is assigned to each train driver during the roster planning process. The Short-Term Data Manager is used to update the train driver information about the realized daily work, absences, unplanned duties and over-time work in the the short-term re-scheduling (a day before the day of operations) by a schedule planner and for the dispatching purposes (on the day of operations) by a train driver dispatcher while the real-time dispatching module is not available.

The automatic scheduling system TURNI (Caprara et al. [1999b]) developed by the Italian company DoubleClick s.a.s. has been used for the automatic generation of the train driver duties at S-tog since 2002. TURNI is used for generating and optimizing the general duty schedules in the tactical planning as well as for strategic analysis purposes. There is an ongoing project in the planning department aimed at using TURNI for special plans during the short-term re-scheduling. Another project includes a performance study of an extended version of TURNI, called weekTURNI, which optimizes a weekly schedule instead of a general daily schedule. Unfortunately, there is no direct interface between PDS and TURNI. The train driver duty schedules produced in TURNI are transferred to PDS by means of the software developed at S-tog.

An Analysis Group of seven operations researchers at S-tog's Planning Department is constantly working on projects regarding possible implementations and analysis of new computer-aided decision support systems on all levels of the planning process (Hofman et al. [2006], Jespersen Groth [2006], Jespersen Groth et al. [2006], Nielsen et al. [2006], Rezanova and Ryan [2009], Nielsen and Christensen [2006], Folkmann et al. [2007], Jespersen Groth et al. [2007], Jespersen Groth [2008b]). Many projects employ Operations Research methods, and several Bachelor (Andersen et al. [2006]), Master (Hofman and Madsen [2005], Villumsen [2006], Føns [2006], Nielsen [2006]) and Ph.D. students (Jespersen Groth [2008a] and this thesis) at leading universities in Denmark are engaged in analysis and prototypes for the decision support at S-tog.

Chapter 2

Railway Disruption Management

The focus in *railway disruption management* is on repairing operations on the railway network when the planned operations need to be adjusted due to disruptions. A *disruption* on a railway network is an event or series of events that cause either some or all operational schedules to deviate from the planned. Railway disruption management concerns modifications to the railway timetable and train routes, as well as to the rolling stock and the crew schedules (including train drivers, conductors and shunting personnel) during and after the disruption. Disruption management decisions are taken on a real-time basis and are referred to as *dispatching* or *recovery* decisions. Dispatching, however, is also the process of monitoring the daily operations, no matter if disruptions occur or not.

In this chapter we present the railway disruption management based on the current practice on the S-train network: timetable, rolling stock and train driver recovery. Some references are given to the publications concerning real-life recovery of trains on the network. An extensive review focused on the crew recovery within the railway and airline industries is presented.

2.1 Disruption Management on S-train Network

2.1.1 Disruptions Classification

The daily operations of S-tog are disturbed by unexpected events almost every day. *Primal* disruption incidents are those which start the disruption in operations. *Secondary* disruption incidents occur as a consequence of primal disruptions. A *knock-on* effect is a propagation of delays over the railway network.

From S-tog's point of view, disruptions on the S-train network can be classified by their source and type. We call the source *internal* if the disturbance is directly caused or can be influenced by S-tog, and *external* otherwise. The type of disruption is *accidental* if it happens suddenly and unpredictably, while a disruption is of a *planning impact* type if it is or could be discovered, influenced or even prevented in advance, for instance during the planning stages of operations. Some disruption causes classified by the the source and the type are given in Table 2.1.

Some disruptions have stronger effect on the operations than others. For instance, a broken signal, which is not repaired for several hours, influences many trains and spreads delays throughout the whole network for many hours. On the other hand, a passenger train departing a couple of minutes later than scheduled from a terminal station of the S-train network can use the buffer time incorporated in the timetable and regain the delay by the time the train reaches the central part of the network.

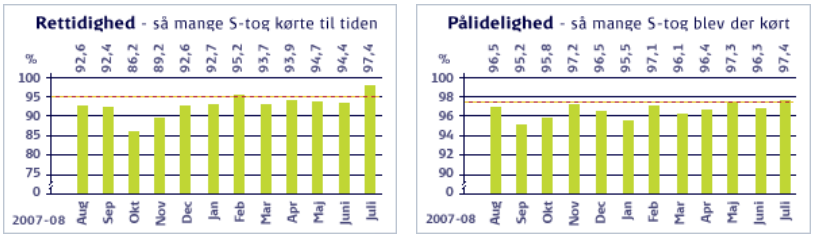
Table 2.1: Disruptions on the S-train network.

Disruption Type	Disruption Source: External	Disruption Source: Internal
Accidental	• Passenger boarding delays • Signalling problems • Weather conditions • Accidents • Obstacles on tracks	• Rolling stock failures • Crew lateness • Dispatchers' mistakes or delayed solutions • Crew mistakes in operations
Planning Impact	• Track maintenance • Seasonal or rush hour changes in demand • Temporary speed limitations, announced in advance	• Lack of rolling stock capacity due to maintenance or minimum cost scheduling • Lack of crew capacity due to tight scheduling • High interdependency between train lines in the timetable

2.1.2 Punctuality and Reliability Measures

No matter what caused the disruption, S-tog performance is measured by two service measures defined in the contract between S-tog and the Danish Ministry of Transport. These are the punctuality and the reliability of services. The *punctuality* is measured by a percentage of the trains arriving on time. An *on-time train* is a train which is not delayed for more than 2 minutes and 29 seconds compared to the scheduled arrival at *any* station of the train line. At present in 2008 the aim is to have 95% of all arrivals on-time every month. The *reliability* of S-tog services is measured by the number of scheduled departures which actually take place. At present in 2008 S-tog' target with respect to reliability is 97,5%. Departure cancellations in the special plans for e.g. track maintenance are not counted in the reliability measure, if the special plan is announced at least 72 hours before the start

of operations. At www.dsb.dk it is possible to follow how the measures are fulfilled for the last 12 months. As an example, Figure 2.1 shows the level of services provided by S-tog between August 2007 and July 2008.



(a) Punctuality (b) Reliability

Figure 2.1: S-tog performance measures from August 2007 to July 2008.

2.1.3 Actors Involved in Dispatch and Recovery

There are three main actors in the dispatching and disruption recovery process on the S-train network: a *network traffic controller* employed by the infrastructure manager Banedanmark, a *rolling stock dispatcher* and a *train driver dispatcher*, both employed by S-tog.

Network traffic controllers (*fjerncontrol leder* in Danish) are employed at Banedanmark. Even though S-tog is presently the only operator on the S-train network, it is Banedanmark that is responsible for the train surveillance and recovery from disruptions on the S-train network, since the infrastructure may be shared by more than one operator in the future. The traffic controllers monitor all train movements on the network using a surveillance system in the network traffic control center. A new network traffic control center DIC-S (*Drifts- og Informationscenter S-tog* in Danish) was opened in 2007. The S-train network is divided into five segments: South (16 stations), West (28 station), City (København H and Dybbølsbro stations), East (27 stations) and North (14 stations), and one traffic controller is responsible for monitoring each segment. Traffic controllers can interfere in the work of switches and signals manually through the signalling system called Speed Control & Train Stop (*HastighedsKontrol & Togstop, HKT*, in Danish). The

network traffic control supervisor (*togleder* in Danish) is the person who is in charge of the overall dispatching decisions on the S-train network.

When a disruption occurs, the recovery process starts with adjusting the timetable by delaying, re-routing or cancelling trains. Recovery strategies on the S-train network are described in more details in Section 2.2.2. As a rule, the recovery decisions are taken in a collaboration with S-tog dispatchers responsible for the rolling stock schedule and the train driver schedule. A rolling stock dispatcher is situated in the same room as the traffic controllers at DIC-S and is responsible for recovering the rolling stock owned by S-tog. A train driver dispatcher is responsible for recovering the train driver duties when they become infeasible. The train driver dispatcher is situated at the main crew depot at København H. Information about disruptions and recovery decisions made by the network traffic controllers is received by the train driver dispatcher by a telephone from DIC-S. The train driver dispatcher can also receive information about disruptions directly from train drivers by a telephone or personally when the drivers arrive at the crew depot. The train driver dispatcher is strategically placed at the crew depot instead of being situated at DIC-S, since many re-scheduling decisions have to be negotiated with train drivers available at the crew depot. The rolling stock and the train driver recovery methods at S-tog are described in Sections 2.3.1 and 2.4.1, respectively.

2.2 Train Timetable Recovery

2.2.1 Train Conflict Resolution

In an undisturbed timetable, each train has a predetermined route, i.e. an itinerary (or a path) through the train network with specified departure and arrival times. A section of tracks between two signals in the train network is called a *block section*. In any feasible timetable, there can only be assigned one train to every block section of the network at any given time in order to avoid train collisions. A significant amount of research is done within the area of strategic, tactical and operational train scheduling, routing and

timetabling on different types of railway networks. An exhaustive review of problems and solution methods related to allocating track sections to trains can be found in Lusby [2008], while a broader overview of railway traffic management, including the real-time dispatch is presented in D'Ariano [2008].

Disruptions on a railway network cause conflicts between train routes. A *conflict* occurs whenever two or more trains require the same block section at the same time. The objective of the conflict resolution problem is to find a conflict-free schedule with as little total delay as possible. This is achieved through delaying, re-routing and cancelling trains. The conflict resolution problem in the real-time environment is similar to the scheduling problem (i.e. allocating block sections to trains over time), but the solution methods must be sufficient for producing high quality recovery solutions quickly. The scheduling and re-scheduling of trains in the railway industry is often formulated as a job-shop scheduling problem, where track sections correspond to machines and trains correspond to jobs. The model is initially proposed by Szpigel [1973]. A train route corresponds to a sequence of job operations on several machines. A feasible train schedule on a railway network contains a set of train routes such that any track section (machine) is claimed by at most one train (job) at any given time.

Publications presenting models, solution methods and decision support systems for the real-time disruption management of trains on the network include but are not limited to local search heuristic approaches, where conflicts are resolved as they appear according to some local criteria (Cai et al. [1998], Şahin [1999], Jacobs [2004]), metaheuristics or other approaches exploring a wider neighbourhood of solutions (Adenso-Díaz et al. [1999], Ping et al. [2001], Wegele and Schnieder [2005], Törnquist and Persson [2005], Törnquist [2007]), exact or semi-exact approaches, where an optimal or near optimal solutions for minimizing delays are found (D'Ariano et al. [2007], Törnquist and Persson [2007]), constraint programming approaches (Rodriguez [2007]) and others.

Two other research areas connected to the railway timetable disruption management are relevant to mention. *Robust planning* is a pro-active method to deal with disruptions. It is focused on methods for generating schedules which can either be able to absorb disruptions or be suitable for a quick recovery with few modifications. For details and references to robust timetabling

and train routing please refer to e.g. Caimi et al. [2005], Vromans [2005], Liebchen and S.Stiller [2006], Herrman [2006] or Kroon et al. [2007], among others. *Delay management* is focused on on re-timing departures of connected trains such that the overall passenger delays are minimized. If a passenger wants to change from the delayed train i to another train j, but misses the connection due to the delay, he/she may have to wait a long time before the next train going toward the required destination arrives. By delaying the departure of the train j, the waiting time for the passenger would be reduced. Delay management can be applied on all levels of the railway planning process, including operational. For details pleased refer to e.g. Schbel [2007], Ginkel and Schbel [2007] or Heilporn et al. [2008], among others.

2.2.2 Recovery Strategies on S-train Network

Network traffic controllers at Banedanmark use different strategies for recovering disrupted operation on the S-train network. These strategies are with various levels of details described in Hofman and Madsen [2005], Hofman et al. [2006] and Jespersen Groth et al. [2007]. They can be classified in three main groups by the severity of the disrupted situation.

Using Timetable Slack

This strategy usually deals with minor disturbances, for instance delays caused by passenger boarding. The network traffic controllers would try to restore order on the network by using extra time margins (slack time) built into the S-tog timetable. The timetable slack can either be built in as the running time supplements or the buffer times. A *running time supplement* is the difference between the scheduled running time and the technically minimum running time of a train. A *buffer time* is the surplus to the turnaround time at terminal stations between train rides. The following actions can be used for this strategy type: delaying trains, reducing running times, reducing headways, reducing dwell times at terminal stations.

Extensive Re-Scheduling

If a disrupted situation is severe, more radical measures have to be taken. Extensive re-scheduling involves re-routing and/or cancelling trains. A train can be re-routed by turning the train before it reaches a terminal station, by overtaking a delayed train at stations with available tracks, by changing the stopping patterns of the trains by skipping stations or swapping the identity of the trains running on the same network segment by turning a fast line into a slow line and vice versa, or by inserting replacement trains at train depots to cover the route of a delayed train and then taking out the delayed train at the same depot. In order to increase the residual capacity on the network even further, certain train tasks between two terminal stations can be cancelled. During the daytime hours, when all network segments are covered by more than one train line, a whole train line or several train lines can be cancelled for a certain period of time or until the rest of the day. Train units assigned to the cancelled trains are shunted to the rolling stock depots along the lines, with the largest depot being at København H.

Emergency Plans

There are approximately 110 emergency plans, which are specially developed to use in different severe disrupted situations at different sections of the S-train network. As an example, a disrupted situation in form of a blockage of a large track segment as a consequence of a broken switch or a suicide accident can be recovered in the same way as other blockages on the same part of the network, by delaying, re-routing or cancelling specific trains. Emergency plans are very useful for the network traffic controllers. Jespersen Groth et al. [2007] give an example of an emergency plan in the timetable-2007 applied to a disruption, where the track section between Dyssegård and Buddinge stations is completely blocked in both directions. The plan involves the description of how the trains of three involved train lines are re-routed and cancelled, and the necessary turnaround times for remaining trains and the required number of train units for implementing the emergency plan.

2.3 Rolling Stock Schedule Recovery

2.3.1 Rolling Stock Re-Scheduling at S-tog

S-tog operates with self-propelled electrical *train units*. A train unit is a module of several passenger carriages with a passage through all carriages. There is a train driver cabin in both ends of a train unit, so the train unit can move in both directions. S-tog rolling stock contains 31 Litra SE train units (4 carriages in the module, shown in Figure 2.2) and 104 Litra SA train units (8 carriages in the module, shown in Figure 2.3). Train unites can be coupled and uncoupled to form trains of different lengths, called *train compositions*, with the longest composition of either two SA units or two SE and one SA units due to the platform length limitations on the S-train network.

Figure 2.2: Train unit Litra SE, length 42.58 meters

Figure 2.3: Train unit Litra SA, length 83.78 meters

Every train unit in a train composition is assigned a line of work (a route) which starts and finishes at a specific depot and covers a set of train tasks during the day of operation. The position of the train unit in a train composition is significant for train units routing, since uncoupling and coupling of units at S-tog's depot stations can only happen from one end of the train. The planned rolling stock schedule has to be *balanced*, i.e. it has to be ensured that the rolling stock inventory at the end of the day at every train depot is sufficient to cover the required rolling stock inventory at the beginning of the next day. Moreover, the operational maintenance requirements are incorporated in the rolling stock schedule. Each train unit at S-tog has to undergo a routine maintenance control every 20.000 km or every 60 days, whichever occurs first. It has to be ensured that each particular train unit is available at

the maintenance depot near Tåstrup station in due time for the maintenance check. The rolling stock schedulers and dispatchers use the rolling stock scheduling system MSS (*Materiel Skedulering System* in Danish), developed at S-tog, for monitoring and scheduling of the train units.

The planned rolling stock schedule often needs to be adjusted due to e.g. track maintenance work on different parts of the S-train network, predictable seasonal changes in the passenger demand or special events, like a football match. Moreover, during the day of operation disrupted situations can make the rolling stock schedule infeasible. The rolling stock dispatcher then re-schedules routes of involved train units in order to reach the overall feasibility of the rolling stock schedule. The the passenger demand on train lines needs to be covered by sufficient number of train units, and the inventory level at the rolling stock depots at the end of the day needs to be balanced. MSS is used for manual adjustments in the rolling stock schedule, but there is no automatic decision support system available yet. For more information about the rolling stock recovery at S-tog see Jespersen Groth [2008a].

2.3.2 Operations Research in Rolling Stock Recovery

There is a very limited number of operations research publications related to the rolling stock re-scheduling and recovery. The first two reviewed papers present research conducted for NS, the passenger division of the Netherlands Railways, while the two last papers are from DSB S-tog A/S.

Budai et al. [2007] deals with a Rolling Stock Balancing Problem (RSBP), which is concerned with eliminating deviations from the desired rolling stock inventory (called the *off-balances*) at certain stations in the otherwise feasible rolling stock schedule. The off-balances can occur when train units end up at other stations than planned due to minor changes in the timetable caused by e.g. track maintenance work or by disruptions on the train network. Two proposed iterative heuristic approaches, which minimize the number of off-balances, perform well even for large instances and can be used both for the re-scheduling and the recovery of the rolling stock off-balances.

The winner of the 2008 INFORMS Railway Applications Section Student

Paper Contest Nielsen [2008] propose an abstract framework for the operational rolling stock recovery decision support system. The rolling stock recovery problem (RSRP) occurs when the current rolling stock schedule is infeasible due to e.g. disruptions or other changes in the timetable. The problem is defined for a certain time period and can be solved with the rolling time horizon, updating information about changes in the timetable every certain time interval. RSRP is formulated as a mixed integer programming model from Fioole et al. [2006], defined over a ceratin limited time horizon and taking disruptions into account, and is solved with CPLEX 10.1 (www.ilog.com) using parts of the timetable and the rolling stock schedule of NS. The short running times (from a few seconds up to one minute) make the solution approach usable in the real-time environment.

Jespersen Groth et al. [2006] present a mixed integer model for re-insertion of a cancelled train line back to operation. The re-insertion problem is specific for the irregular operations of S-tog and occurs when all train departures of one or more train lines are cancelled for a certain time period as a consequence of a recovery strategy on the S-train network described in Section 2.2.2. The problem deals with inserting previously withdrawn train units from several depots back into operation such that *all* train departures which follow the start of the re-insertion process are covered. The model is implemented in GAMS (www.gams.com), and optimal solutions to all possible re-insertion scenarios are calculated. Solutions are incorporated in a user interface, which is used by S-tog rolling stock dispatchers in the real-time environment.

Jespersen Groth [2008b] presents a decomposition approach to the rolling stock recovery problem at S-tog. The RSRP is decomposed into a composition part and a routing part. The composition part is modelled with the Train Unit Position Model, a mixed integer program which finds the optimal number of train units of different types (Litra SA or Litra SE) and their positions in the train for each train task in the recovery problem with respect to minimizing a weighted sum of following objectives: the seat shortage, the number of composition changes (coupling or uncoupling of train units), the cost of covering train tasks with train units and the difference to the originally scheduled depot capacity. The routing part is expressed by two models. The Train Sequence Model is an assignment model with side constraints. It solves a pre-assignment of those train units which are sufficient for covering an entire train sequence (a route) alone. Since it is not necessary to take

care of composition changes in the Sequence Model, the problem is relatively easy to solve. The Train Routing Model assigns the train units which are not assigned to train tasks in the previous model. All three models are implemented in C# and solved with CPLEX 10.0 Concert Technology. Extensive experiments for defining and validating the weights in the Position Model and testing how the presence of the Sequence Model affect the performance of the Routing Model are presented. The average solution times are somewhat high for the real-time implementations, and further refinement of the models and solution methods is required.

2.4 Train Driver Schedule Recovery

2.4.1 Train Driver Schedule Recovery at S-tog

Figure 2.4 shows the current process associated to the train driver schedule recovery at S-tog. As described in Section 2.1.3, all communication with the network traffic control center and train drivers is conducted over the telephone. The short-term data manager of the crew scheduling system PDS is used by the train driver dispatcher to monitor the train driver schedule. The system has good drag-and-drop functions, and the dispatcher is able to view and alter the train driver schedule by removing or adding train tasks in the duties. One of the disadvantages with using the short-term data manager module of PDS for the dispatching purposes is that it is not intended for the real-time dispatch and recovery. The time for uploading every duty change to the system is therefore simply too long, and the train driver dispatcher often implement recovery solutions to operations without updating the system. Then, at the end of the day or when the situation on the S-train network does not require dispatcher's constant attention, the changes to duties are entered into the system. It is important to upload changes to the train driver duties in order to register evt. overtime and absences. The train driver dispatcher can also use PDS for trying out different recovery scenarios without updating the system. It is planned to install the real-time dispatching module of the system and use it for dispatching purposes instead of the short-term scheduling module from year 2009. PDS is neither interconnected with the

network traffic control system at DIC-S nor with the rolling stock scheduling system MSS, which is used for monitoring the train units.

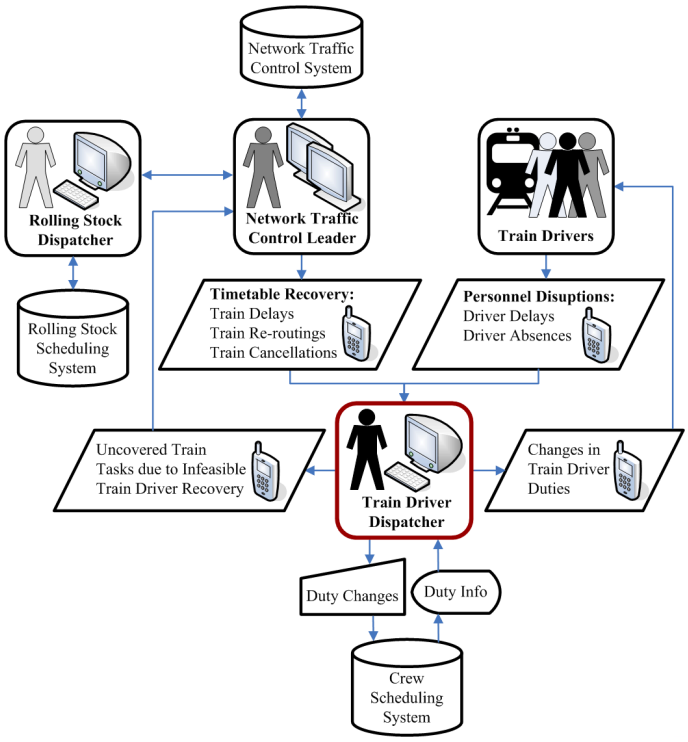


Figure 2.4: Train driver recovery process at S-tog.

When a train driver dispatcher receives a message from the network traffic control supervisor or a train driver about a certain disruption and/or a certain timetable recovery strategy, he or she has to find out what impact the particular disruption has on the train driver schedule. An experienced train driver dispatcher can immediately identify the duties affected by delays, re-routings or cancellations of trains. A train driver schedule is *disrupted* when at least one connection in at least one train driver duty is broken, and, as a consequence, the driver is not able to perform the subsequent task scheduled in the duty. Consider a small example of a disruption. Figure 2.5 shows a part of the timetable-2007 with three scheduled train driver duties.

Let us assume that at 4 p.m. the train driver dispatcher receives following

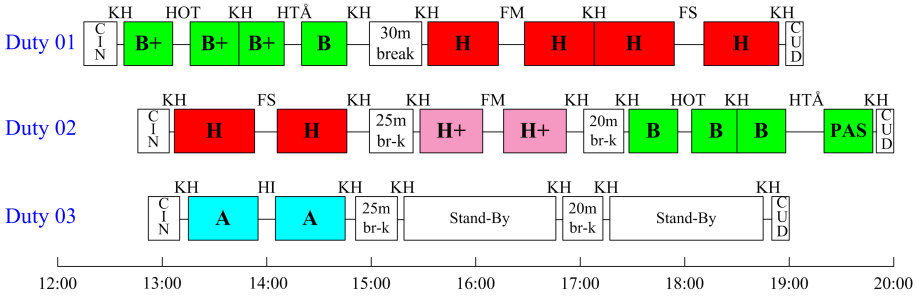


Figure 2.5: Before the disruption.

information: the train of line **H** arriving to Farum station (abbreviated FM) is 15 minutes delayed. The network traffic controller reduces the dwell time of the train at FM in order to recover the subsequent departure of line **H**, and the train can depart from Farum station with only 3 minutes delay compared to the scheduled departure. In order to increase the residual capacity on the S-train network, the network traffic controller cancels line **H+**, which runs on the same segment of the network as line **H**. Hence, the train departure of line **H+** from FM to København H (abbreviated KH) is cancelled. This disrupted situation is shown in Figure 2.6.

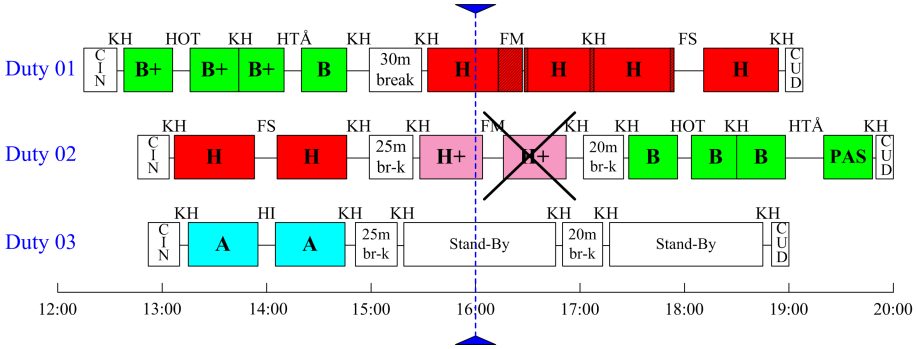


Figure 2.6: Disruption example.

When a train driver is not able to cover a particular train task in the duty, the train driver dispatcher tries to find another driver to cover the train task. Most often it is a reserve driver who is on stand-by at a crew depot. Sometimes it is necessary to make swaps between train tasks within two or

three driver duties. These actions require concentration and experience from the dispatcher. An experienced dispatcher can most often quickly solve the puzzle of recovering the train driver schedule. However, in some situations the impact of disruptions on the train driver schedule is so severe, that it is simply not possible to find a train driver for all non-covered train departures. In those cases trains have to be delayed or cancelled.

The work of the train driver dispatcher often requires a lot of self-control, persuasion abilities and a good relationship to the train drivers, since it is sometimes necessary to ask a particular train driver to cover a task which slightly violates the duty rules. Recovery decisions are based on previous experience, and the quality of each recovery schedule depends on the particular train driver dispatcher. Re-scheduling decisions have to be made quickly under tremendous pressure and a lot of time is used for communicating decisions to the drivers.

In the given example the train driver dispatcher can see that Duty 1 is not yet disturbed to an extent where it has to be recovered, since it is feasible for the train driver to make a connection between the arrival of line **H** to Farum and the subsequent departure of the line. Duty 2 is on the other hand disrupted, since the driver cannot return to KH with the scheduled train task of line **H+** due to the cancellation of the line. The dispatcher contacts the train driver assigned to Duty 2 and asks her/him to ride as a passenger on the train of line **H** back to KH, where the driver has to take a break according to his/her duty schedule. This part of a duty recovery is shown in Figure 2.7.

Duty 2 is however not fully recovered yet. The passengering train driver will be 13 minutes delayed for the meal break. Since the driver is entitled to a 20 minutes break according to the duty schedule, the driver will not be able to cover the next train task on line **B** from KH to Holte station (abbreviated HOT). In some situations it is possible to shorten the scheduled short break of the driver, if the second short break can be prolonged and the overall break rules described in Section 1.4.1 are satisfied in the duty. However, in the present situation the driver has already held a 25 minutes break, hence the time of the second break in the duty cannot be shortened. In order to recover the situation, the train driver dispatcher assigns a half-trip on line **B** from KH to HOT and back to KH to another driver assigned to Duty 3,

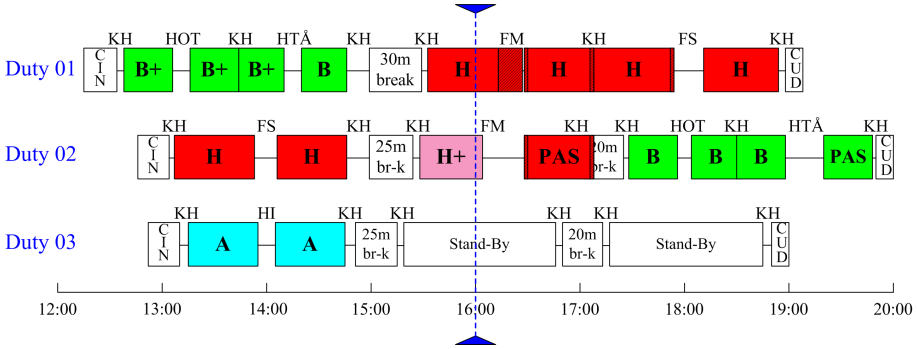


Figure 2.7: First step in recovery: passengering task.

who is on stand-by at that moment. Taking the half-trip to HOT will not make the duty of the reserve driver infeasible, since the driver can be present for the scheduled check-out at KH crew depot in due time. The particular disrupted situation is recovered in two and a half hours. Given that no other disruption occurs, all three train drivers return to their original duties at the end of the recovery.

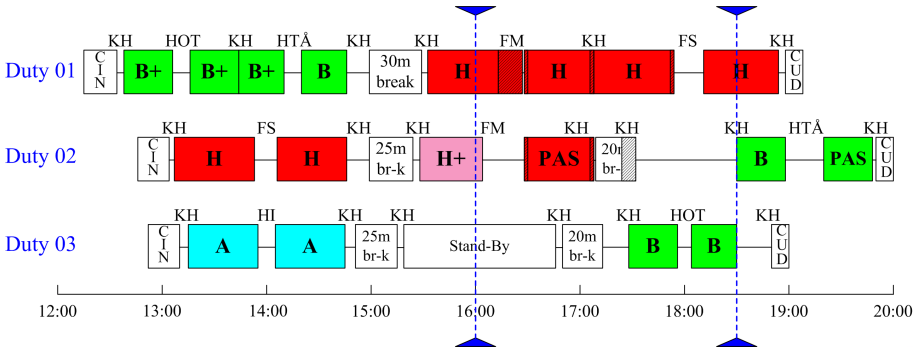


Figure 2.8: Recovery from the disruption.

2.4.2 Operations Research in Railway Crew Recovery

Crew disruption management within the railway industry has not been given much attention by operations researchers yet. Apart from the publications

related to this thesis (Rezanova and Ryan [2006], Rezanova and Ryan [2009]), only three applications within the railway industry related to the present research have appeared in the literature.

An integer programming approach to a simultaneous train timetable and crew roster recovery problem is presented in Walker et al. [2005]. Starting with an integrated train and driver scheduling model, the authors develop an approach for disruption recovery in real-time. The objective of the model is to minimize the deviation from the existing schedule, while minimizing the cost of the adjusted crew shifts. Limiting the time period for which the schedule is resolved, the problem size is kept small enough to produce optimal solutions fast. The problem is modelled as a set partitioning problem with additional constraints, which ensure the time consistency of the pieces of work within driver shifts and the maximum duration of shifts. The problem is solved with a branch-and-bound algorithm with column and constraint generation. The linear programming relaxation of the problem is solved using the primal revised simplex method. Hot-starting from a solution, which was feasible prior to the disruption, the optimization starts with pricing out variables, which represent pieces of work and idle times for train tasks. When the train variables are priced out, the driver shift variables are considered, constructing columns from initially constructed potential driver shifts, inserting a meal break into each shift. Fractions are resolved using constraint branching. Illegal train crossings and overtakes of trains are resolved by adding constraints as these are needed. The model is tested on a one day timetable for the Wellington Metro line in New Zealand, covered by 36 trains. The train services were split into 564 pieces of work at possible relief points. Delays of different duration were introduced on 3 trains in order to disrupt the schedule. Optimal solutions were produced in reasonable time, ranging from about 26 seconds to under 2 minutes on Pentium 200 MHz computer with 64 MB RAM.

Huisman [2007] presents a solution to the Crew Re-Scheduling Problem for train driver duties disrupted due to the maintenance work on train tracks. Data from NS Reizigers is used to test the model. The model is formulated as a set covering problem, allowing each train task to be covered more than once, representing the deadheading of the crew. For each original duty, a large set of "look-alike" duties is generated, replacing parts of the original duties with different pieces of work. The Lagrangian relaxation of the model is

solved with column generation, pricing out "look-alike" duties with negative reduced cost. If the set of "look-alike" duties is not sufficient to find the optimal (or near optimal) solution, other duties with negative reduced cost are generated as simple shortest paths on the directed graph, where each vertex represents a piece of work, i.e. the sequence of train tasks on the same rolling stock. The problem is solved to integrality using the heuristic approach for the set covering problem of Caprara et al. [1999b] implemented in the crew scheduling package TURNI (www.turni.it). Test results from two real-life cases are presented. The first case with 586 duties and 5,683 train tasks generates 8,767 "look-alike" and 18,4420 other duties. The problem is solved within 9 CPU hours on Pentium IV computer (3 GHz, 512 MB RAM). The second case with 773 duties and 7,740 train tasks generated 16,9974 "look-alike" and 20,3961 other duties. The problem is solved within 16 CPU hours. While the solution method is applicable for a short term rescheduling and is successfully employed at NS Reizigers, the solution times are too long from the operational recovery decision support system point of view.

Potthoff et al. [2008] consider the operational train driver rescheduling problem at NS. Given a disrupted timetable, the objective is to find a new crew schedule that covers as many train tasks as possible. A core problem is defined with a limited number of train drivers included in the problem, and the solution is aimed to be achieved within a certain recovery period. The problem is a set covering formulation of the crew scheduling problem with additional variables, which explicitly determine if a certain train task is cancelled or not. The set covering problem is solved using a Lagrangian heuristic similar to the one proposed in Huisman [2007], improving the heuristic with partial pricing, early termination and column fixing. First, an initial solution to the core problem is computed. If some train tasks are still uncovered, a neighbourhood exploration is performed by adding other drivers and train tasks assigned to their duties. A driver is added to the set of candidate drivers, if he/she can possibly cover the uncovered tasks. Moreover, a set of replacement drivers which can possibly swap tasks with the candidate drivers are also added to the problem. The solution is implemented in C++. Experiments are run on an Intel Pentium D processor (3.4 GHz, 2 GB RAM). Ten test scenarios are based on adjusted historical disruption events, with the 3 hours recovery periods, where between 15 and 59 driver duties are affected. Three sets of experiments are based on different initial core problems, one

without reserve drivers, and two with 46 and 20 reserve drivers, respectively, resulting in 30 test instances. Initial core problems contain from 31 to 117 drivers without including reserve and between 186 and 658 train tasks. The computational times for solving the initial core problems span from 7 to 181 seconds. Further experiments were made with expanding the core problems of those 8 out of 20 instances, where some tasks were not covered (at most 2 tasks were left uncovered). Neighbourhood explorations reduced the number of uncovered tasks, but not in all instances. A neighbourhood exploration run took between 6 and 392 second, dependent on the instance and the size of the neighbourhood.

2.4.3 Airline Crew Recovery

Due to a very limited operations research literature on the train driver recovery and re-scheduling solutions, it is inevitable to expand the background search to other transportation industries where similar workforce recovery problems occur. The airline crew recovery has been a subject of research since the mid 1990's. The research within airline disruption management started in the mid 1980's, focusing on the aircraft recovery. For a resent review of disruption management in the airlines, including aircraft, crew, passenger and integrated recovery, please refer to Clausen et al. [2009]. We present a compact literature review of the published work within the airline crew re-scheduling and recovery in Appendix A.

A common assumption in the majority of the research within the airline crew recovery is that the flight schedule and aircraft rotations are recovered before the crew re-scheduling decisions are made (Wei et al. [1997], Stojković et al. [1998], Guo [2005a], Nissen and Haase [2006], Medard and Sawhney [2007]). This view is supported by the hierarchical recovery process, which is followed in the airline operations control centers. However, a decision for cancelling a flight are not taken unless it is ensured that there is an aircraft and a crew available for covering the consecutive flights. When the flight schedule is fixed, the crew recovery problem can be formulated as a tactical crew pairing or rostering problem. Other authors extend the classical crew scheduling formulation of the recovery problem by adding a set of decision variables, which allow to cancel flight legs (Johnson et al. [1994], Lettovsky et al. [2000],

Yu et al. [2003]). Finally, problem formulations of the crew recovery problem, which explicitly account for departure delays, are reported in Stojković and Soumis [2001], Abdelghany et al. [2004], Stojković and Soumis [2005] and Zhao et al. [2007]. Each minute of departure delay is given a cost in the objective function, while the flight precedences and delay limitations are ensured by constraints in the models.

All researchers agree that airline and the railway crew schedules are far too large to re-optimize globally in real-time, hence only a shapshot of the entire system within a certain time window is re-scheduled. Another important feature of the recovery problems compared to the planning problems is that feasible solutions are more important than optimal solutions because the real costs of modified duties are difficult to estimate when the crew members are payed fixed monthly salaries. However, it is important to consider the additional costs for deadheading, overtime payments etc.

Airline crew recovery problems have typically a different time horizon than the railway crew recovery problems on dense networks with the focus on the passenger transportation during the daytime, such as the S-train network or the Dutch railway network (Potthoff et al. [2008]). An airline crew pairing usually lasts a few days and therefore includes one or more overnight. Rest time requirements for pairings are often complex and very strict, and they have to be considered in the airline recovery solutions. The situation is different when the daily duties are re-scheduled. As an example, duty rules regarding the time spent away from the base are not necessary to consider when re-scheduling the train drivers on the S-train network. Apart from the pairing requirements, the individual rostering constraints are considered in some airline recovery problems (e.g. Medard and Sawhney [2007]). This complicates the recovery problem even further.

The solution space for the crew recovery problem is often limited to resolving the most urgent conflicts in the schedule. Along with the increasing computer power, solution methods applied to the crew recovery problem has evolved during the last two decades from simple heuristic methods to more complex optimization approaches. However, since the solution space of the recovery problem is limited to a certain part of the schedule, the overall solution to the recovery is not proven to be optimal, even when exact optimization methods are applied.

Chapter 3

Solution Framework

A framework for the train driver recovery problem (TDRP) is presented in this chapter. We solve TDRP instances on a rolling time horizon, while keeping the problem space of TDRP as small as possible, only expanding it when necessary. The key concepts of the framework are: the disruption neighbourhood, the recovery duty, the recovery period, and the disruption neighbourhood expansion. The solution framework is generic and can, with minor modifications, be used in the operation of any passenger train operator. The framework is also applicable in combination with solution approaches to crew recovery problems different from that which is presented in this thesis.

A prototype for a train driver recovery decision support system (TDR–DSS) is presented in this chapter. The required data input to the TDR–DSS is described, and the current data format used in the prototype to the TDR–DSS is presented. We also propose an information flow between the system and other actors involved in the recovery process at the S-train network. Finally, the graphical user interface of the prototype is presented.

3.1 Recovery Objectives

An ambitions operations researcher would claim that the objective of the TDRP is **to return to the original train driver schedule as soon as possible by robust and minimum cost re-scheduling of the train driver duties**. However, a realistic train driver dispatcher's primary objective for a recovery is **to have a driver present at as many departures as possible by any means**.

As described in previous chapters, the recovery decision making at S-tog has a hierarchical structure, and the train driver schedule is often the last resource that is altered once recovery measures for the timetable and the rolling stock plan have been taken. For this project we therefore focus on recovering the train driver schedule as a stand-alone application without making an attempt to recover the whole operation in one integrated manner.

The cost of the recovery is not determined by a physical cost of the driver schedule (the drivers are already paid to be at work), but rather by the fictitious cost which expresses how unattractive each recovery duty is. The *optimality* of the solution is hence not as important as the *feasibility* of the solution. Furthermore, it is more important to ensure feasibility of the schedule in the very near future than to re-schedule driver duties several hours ahead, since other disruptions are likely to occur over such a time horizon.

Another objective is to modify the train driver schedule as little as possible. Moreover, S-tog train driver dispatchers aim at disturbing as few train driver duties as possible, i.e. two disturbed connections in one driver's duty is more preferable than one disturbed connection in each of two drivers' duties. The central part of the S-train network contains the busiest stations in the whole network, where even a small departure delay can cause a knock-on effect to the subsequently arriving trains. The train driver dispatchers therefore prefer to avoid train driver changes from train to train at København H.

In the general sense, the word *recovery* means to regain or retake the possession of or to return something back to its original state. In order to justify the train driver recovery problem name as opposed to the train driver *re-*

scheduling, the goal of the train driver recovery is to return to the original train driver schedule as fast as possible after a disruption occurs. An important condition for satisfying this objective is that the train timetable and the rolling stock schedule are back to the original state at the end of the train driver recovery. This is unlikely for large disruptions. However, the solution method relies on a continuous recovery with a rolling horizon, as explained further in this chapter. This allows the train driver schedule to be modified again and again until the operation on the S-train network gets back to the normal state or the day of operation ends.

3.2 Key Concepts of the Framework

3.2.1 Disruption Neighbourhood

When a disruption occurs, it does not instantly affect the whole timetable or the whole train driver schedule. In the beginning of a disruption only a few driver duties are disturbed. As time progresses, more and more train tasks, and hence more and more train driver duties, can be affected by the same disruption.

For a particular disrupted situation we identify a *disruption neighbourhood*, which contains a number of train drivers and a number of train tasks. The number of train drivers and train tasks within the disruption neighbourhood is relatively small compared to the daily train driver schedule. Every train driver within the disruption neighbourhood is to be assigned a *recovery duty*, which is a sequence of activities assigned to a train driver as a substitute of part of his/her original duty. A recovery duty contains either a subset of train tasks within the disruption neighbourhood or does not contain any train tasks, corresponding to the driver spending the time in reserve.

For each train driver in the disruption neighbourhood a commencing and a terminating task are determined, defined by the length of the *recovery period* as explained further. The recovery period is the time span between the recovery start time and the recovery end time. The duration of the recovery

period determines the size of the disruption neighbourhood, i.e. the longer the recovery period, the larger the number of train tasks and train drivers included in the disruption neighbourhood. A *commencing task* is the last task assigned to the train driver in the original duty immediately before the recovery start time or the first task of the driver's original duty, if the duty starts within the recovery period. The train driver has either just finished or is still performing the commencing task at the beginning of the recovery period. A *terminating task* is the task that follows the last task of the driver's recovery duty, if the original duty continues after the end of the recovery period, or the last task in the recovery duty, if the original duty ends within the recovery period. The commencing task of a reserve driver is a stand-by task which ends at the recovery start time. The terminating task of a reserve driver is always the check-out task of the original duty. All train drivers in the disruption neighbourhood are required to return to their terminating tasks at the end of the recovery duties. Computational experiments show that the best recovery solutions are achieved if terminating tasks of train drivers are limited to the tasks which begin at crew depots. However, in order to demonstrate the disruption neighbourhood and different expansion strategies, the train driver recovery duties in the presented examples can finish at any terminal station.

The *initial* disruption neighbourhood for a given disruption is generated in the following way: First, a set of train tasks which are disrupted within the recovery period is collected. A train task is *disrupted* if it is delayed, cancelled, re-routed or uncovered, i.e. assigned to a driver who is unavailable for the departure. The collection of disrupted train tasks is based on the available information about the disruption. The information can contain the knowledge about actual and expected events which are the consequence of the disruption. Information about either already delayed trains or the trains which are expected to be delayed is available from the network control center. Re-routings and cancellations of trains are usually determined for some time in the future, when decisions about recovering operations on the S-train network are made by the network traffic controllers. Information about acutely absent drivers is reported directly to the train driver dispatcher. Second, when the disrupted train task are determined, all train drivers assigned to these tasks in the original train driver schedule are included in the initial disruption neighbourhood. For each driver the commencing and the terminating train tasks are determined. Finally, all train tasks from the original driver

duties between the commencing and the terminating train tasks are included in the train task set of the disruption neighbourhood. If any meal breaks are scheduled in the drivers' original duties between the commencing and the terminating tasks, the start times and the lengths of the breaks are collected. As an example, the initial disruption neighbourhood to the disrupted situation described in Section 2.4.1 is presented in Figure 3.1. The disruption neighbourhood is defined for a recovery period of two hours and contains two train drivers, assigned to Duty 1 and Duty 2, who are immediately affected by the disruption, and three train tasks, line **H** (Farum–København H, København H–Frederikssund) and line **B** (København H–Holte). When a feasible recovery solution is found within the initial disruption neighbourhood, the objective of disturbing as few train driver duties as possible in the recovery is achieved, since *all* initially involved driver duties are affected by disruption, and hence need to be modified.

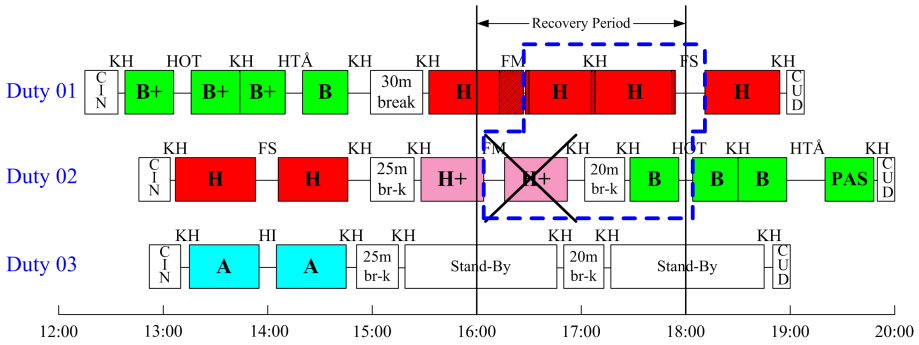


Figure 3.1: Initial disruption neighbourhood.

The length of the recovery period is important for the quality of the recovery solution. The longer the recovery period, the larger the time span between the commencing and the terminating task of each driver and the larger the number of disrupted and non-disrupted train tasks that are included in the disruption neighbourhood. Hence, more possible recovery duties can be generated within the disruption neighbourhood, and there is a better chance for achieving a feasible solution. If the recovery period is too short and only a very limited part of each disturbed driver duty is considered, it can be impossible to find a feasible recovery solution within the initial disruption neighbourhood. On the other hand, the computational time for a finding recovery solution increases with the number of possible recovery duties to be

considered in the optimization. Very long recovery periods are therefore not appropriate, since the short response time is important when a decision has to be made. A natural trade-off between the quality of the solution and the computation times therefore occur in some problem instances. In any case, there is little reason to find a recovery solution for the train driver schedule several hours ahead. The scheduling solution at the end of the long recovery period will most probably be useless as the exact information about disruptions further in time is not available, and more disruptions are likely to occur in the meantime.

3.2.2 Expansion of Disruption Neighbourhood

If a feasible solution to the train driver recovery cannot be found within the initial disruption neighbourhood, the disruption neighbourhood is *expanded* by either adding more drivers and/or extending the recovery period. Consider the disruption neighbourhood example in Figure 3.1. The train driver assigned to Duty 2 has to be available at Holte station (HOT) at the end of the given two-hour recovery period. The same situation is described in the manual recovery example in Section 2.4.1. Even if the driver is not assigned to any train tasks within the initial disruption neighbourhood, no feasible recovery duty could be generated for the driver. There is simply not enough time to travel from the arrival station of the commencing task, Farum station (FM), to the crew depot at København H (KH) to hold the scheduled meal break, and then to travel to the departure station of the terminating task. In a situation where a feasible recovery duty cannot be generated for at least one train driver in the disruption neighbourhood, the recovery period for the affected driver is expanded with a certain period of time, e.g. 30 minutes. The *recovery period expansion* is illustrated in Figure 3.2.

After the recovery period expansion for the train driver assigned to Duty 2 the disruption neighbourhood contains two train drivers and four train tasks, the newly added task being the train of line **B** from HOT to KH. Due to the meal break constraint of the driver assigned to Duty 2 it is still not possible go generate a feasible recovery solution. The driver assigned to Duty 1 can cover the half-trip of line **B** after driving the train from FM to KH, but then the train task of line **H** from KH to Frederikssund station (FS) is not covered.

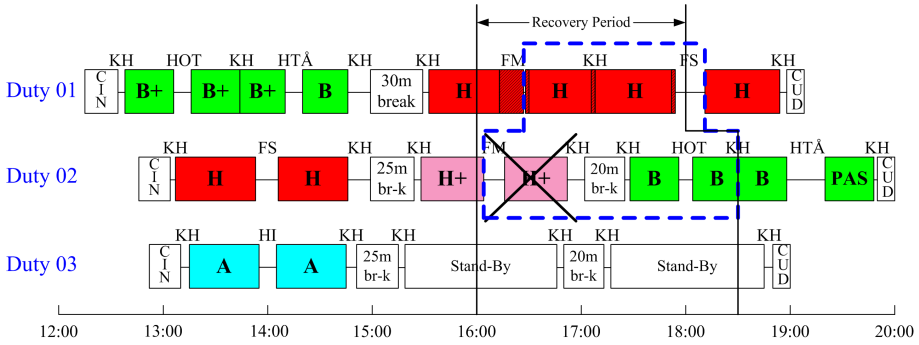


Figure 3.2: Recovery period expansion for Duty 2.

In this situation it is necessary to expand the disruption neighbourhood by *adding train drivers* to the problem. The obvious choice here is to add one or more reserve drivers. The disruption neighbourhood expansion by adding one reserve driver is shown in Figure 3.3. In Section 5.7 it is explained how the disruption neighbourhood expansion is handled in the solution process.

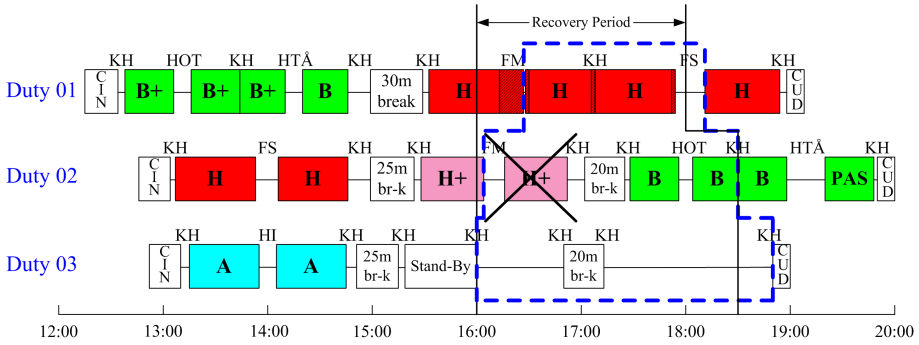


Figure 3.3: Adding a reserve driver to the disruption neighbourhood.

3.2.3 Rolling Time Horizon Recovery

A few minutes after the solution to a particular instance of the TDRP is found, a new disruption may occur or new information about the impact of the current disruption may become available. It is therefore very important

that the decision support system for the train driver recovery is constantly updated with new disruption information and the TDRP instances are generated and solved sequentially with a *rolling time horizon* during the day of operation. In other words, we attempt to solve the *dynamic* problem by solving a sequence of *static* train driver recovery problem instances generated with a rolling time horizon.

We define a *monitoring interval* as the period of time, which is considered to be suitable for how often the recovery solutions are generated. The monitoring interval can, for instance, be set to 20–30 minutes. Regardless of how much disruption information is available, it can be used for generating the initial disruption neighbourhood. If no disruption information is available at the time of the system update, no changes are applied to the train driver schedule. The length of the monitoring interval must not be too long, since it is important to evaluate the status of the situation as often as possible.

The continuous recovery process on a rolling time horizon is illustrated in Figure 3.4. The changes to the train schedule caused by the first disruption are applied to the undisturbed schedule, and the first disruption neighbourhood is defined (Part I of Figure 3.4). If the solution to the first TDRP over the initial disruption neighbourhood is infeasible, the disruption neighbourhood is expanded by for example adding other drivers as shown in Part II of Figure 3.4. When a feasible solution is achieved and accepted by a train driver dispatcher, the driver schedule is modified according to that solution. If another disruption occurs, the changes caused by the second disruption are applied to the "new" schedule, which contains the solution to the previous TDRP (Part III of Figure 3.4). The new disruption neighbourhood may or may not include some or all drivers from the previous instance. The new instance of the TDRP is resolved over a new disruption neighbourhood and the train driver schedule is modified accordingly. The recovery process continues as illustrated on Part IV and Part V of Figure 3.4.

In each TDRP iteration, the term *original schedule* refers to the train driver schedule obtained by solving the previous instance of the TDRP or, in case of the first daily disruption, to the planned schedule employed at the beginning of the day. Likewise, the *original duty* of a driver is the duty generated for the driver in the previous instance of the TDRP or the unchanged duty from the undisturbed schedule.

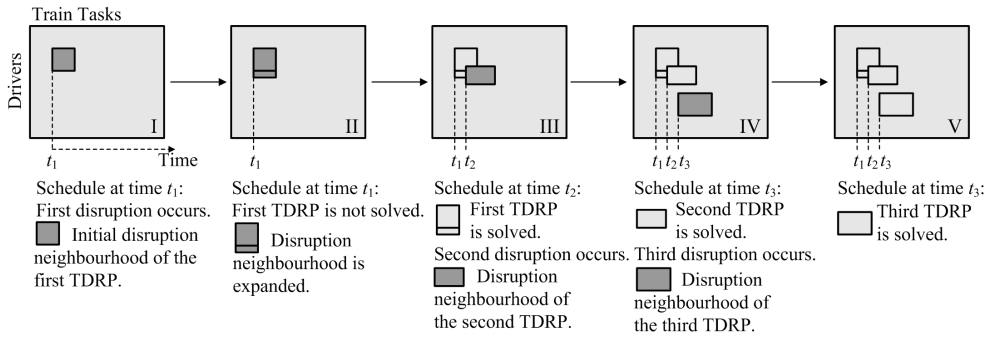


Figure 3.4: The rolling time horizon recovery process.

3.3 Decision Support System Prototype

3.3.1 Data Input to TDR–DSS

The following input data is necessary for generating and solving an instance of TDRP in the current version of the prototype for the train driver recovery decision support system (TDR–DSS) developed during this project: the scheduled daily *timetable*, the scheduled daily *train driver schedule*, and the *disruption data*, which includes changes to the timetable expressed through train delays, re-routings and cancellations, and information about driver delays or absences. Data representations currently used in the prototype are described further in this section. Apart from this information, the train driver duty rules components have to be present in the system. In a real-life implementation of the TDR–DSS it is advisable to include the daily *rolling stock schedule* and lists of changes to the scheduled train unit routes as a consequence of recovery solutions made by the rolling stock dispatcher.

The Timetable

The scheduled timetable is presented by a set of *trains*, each train having a unique train number and a stopping pattern, which is a sequence of stations

with the scheduled departure and arrival times. An example of the stopping pattern for a train number 10100 of line **A** from Hundige to Hillerød station in timetable-2007 is presented in Table 3.1. The complete stopping pattern of this train can be found in Appendix B. An empty entry in the third column means that the train passes the corresponding station without stopping, e.g. train 10100 does not stop at Virum or Sorgenfri stations. The timetable is given in a hh:mm& format. An ampersand (&) indicates that the arrival/departure time takes place later than hh:mm:30, e.g. 23:51& in the timetable stands for 23:51:40, the scheduled arrival of the train at Ishøj station. The official timetable for the passengers includes departure times for the stopping stations only, given in a hh:mm format.

Table 3.1: Example of a stopping pattern.

Station	Abbr.	Arr/Stop Time	Dep/Pass Time
Hundige	UND		23:49
Ishøj	IH	23:51&	23:52
Vallensbœk	VLB	23:54&	23:54&
⋮	⋮	⋮	⋮
København H	KH	00:14:00	00:15:00
⋮	⋮	⋮	⋮
Lyngby	LY	00:33&	00:33&
Sorgenfri	SFT		00:35&
Virum	VIR		00:37
Holte	HOT	00:38	00:38&
Birkerød	BI	00:42	00:42&
Allerød	LI	00:47	00:47&
Hillerød	HI	00:54	

The Train Driver Schedule

The daily train driver schedule is a set of train driver duties. Each train driver duty is a set tasks performed in a sequence, and the undisturbed train driver schedule covers all train tasks in the daily timetable of S-tog. The prototype to the TDR–DSS reads two schedule data representations that are shown

in Appendix C. An overview of the stations on the S-train network with the corresponding abbreviations used by the planners at S-tog is presented in Appendix D. Abbreviations of activities are presented in Section 4.2.1, while the technical connection times are described in Section 1.4.1. To give an example, a part of the duty 209 is shown in Figure 3.2. Briefly, the driver assigned to the duty 209 checks in at København H depot at 11:53, drives a full trip on the train of line **H** (København H–Farum–København H–Frederikssund–København H), has a long break which is scheduled for 30 minutes, but with the slack in the duty it lasts for 35 minutes from 15:37 to 16:42, and continues with the duty until 18:41, when the checks-out finishes.

Table 3.2: A train driver duty representation.

DutyNr	TrainNr	Task Abbr.	StartTime	EndTime	DepStation	ArrStation
209		CIN	11:53	12:08	KH	KH
209		BEV	12:08	12:12	KH	KH
209	50136	PIT	12:12	12:13	KH	KH
209	50136	TFF	12:13	12:52	KH	FM
209		SPD	12:52	12:56	FM	FM
209	50241	TFF	13:08	13:47	FM	KH
209	50241	TFF	13:48	14:32	KH	FS
209		SPD	14:32	14:36	FS	FS
209	50146	TFF	14:46	15:32	FS	KH
209	50146	PIF	15:32	15:33	KH	KH
209		BEV	15:33	15:37	KH	KH
209		PAU	15:42	16:12	KH	KH
209		BEV	16:12	16:16	KH	KH
⋮	⋮	⋮	⋮	⋮	⋮	⋮
209		BEV	18:27	18:31	KH	KH
209		CUD	18:31	18:41	KH	KH

Disruption Data

Disruption data is preprocessed into a list of deviations from the scheduled train stopping pattern and a list of train driver delays and absences, which are independent on deviations to the timetable (e.g. driver delay to a duty check-in or an acute illness). Table 3.3 presents the four disruption data types with the required data entry fields. The four data types are used to express

changes to the timetable and the train driver schedule. The same train task in the schedule can be affected by several disruption data types. For example, a delayed train which is turned before the terminal station is reached for the timetable recovery purposes is described as a combination of departure delays, arrival delays and a partial train task cancellation. Disruption data input can be actual or expected based on the information received during the last monitoring interval, as described in Section 3.2.3. Disruption data representation used for computational experiments is described in Section 6.1, where Table 6.1 presents an example of a disrupted train.

Table 3.3: Disruption data types.

Departure Train Delay	
`[TrainNr, DepStation, NewDepTime]`	A departure delay larger than a certain predefined minimum delay time (e.g. 1 or 2 minutes).
Arrival Train Delay	
`[TrainNr, ArrStation, NewArrTime]`	An arrival delay larger than a certain predefined minimum delay time (e.g. 1 or 2 minutes).
Train Task Cancellation	
`[TrainNr, FromStation, ToStation]`	A cancelled train task or a part of a train task. A part of a cancelled train task is usually connected to a train re-routing, for instance when a train is turned before reaching the terminal station.
Driver Unavailability	
`[DutyNr, FromTime, FromStation, ToTime, ToStation]`	A train driver delay for a train task in the duty due to *other* reasons than the delay of the driver's previous train task. A train driver absence from the duty or a part of the duty.

3.3.2 Information Flow Diagram

An overview of a possible information flow between the train driver recovery decision support system and other actors and systems at S-tog is shown in Figure 3.5. Compared to the current recovery information flow shown in Figure 2.4, the presence of TDR–DSS makes the train driver recovery process automatized to a higher degree than the current process. The information flow scenario presented here is only one of the possible configurations. It requires either a direct communication or an interface between TDR–DSS and the network traffic control system DIC-S, the crew scheduling system PDS, and the rolling stock scheduling system MSS. Furthermore, the suggested scenario allows a direct communication from TDR–DSS to the train drivers via, for example, personal digital assistants (PDA's). However, due to the current safety requirements it is not allowed to communicate vital information to the train drivers using electronic equipment only, e.g., by sending a text message to the train driver's mobile phone. The train driver dispatcher is required to speak to the train driver personally in order to make sure that the message is received correctly and the driver confirms to act accordingly. Therefore, information about changes in train driver duties is sent through two communication channels in the suggested scenario.

On the night before the day of operation the timetable and the train driver schedule are transferred to TDR–DSS from the crew scheduling system PDS, and the rolling stock schedule is transferred from the rolling stock scheduling system MSS. The short-term re-scheduling adjustments to the rolling stock and crew schedules are uploaded to the scheduling systems no later than 12 hours prior the start of operations, i.e. at approximately 3:30 p.m. on the afternoon before the day of operation. Therefore, the schedules due for the next day of operation can be transferred to TDR–DSS immediately after this deadline or, preferably, after the end of the daily operations at approximately 2:30 a.m. the following morning.

During the day of operations the timetable disruption data is supplied from the network traffic control system DIC-S, including train delays, re-routings and cancellations. Information about acute absences or delays of train drivers is received by the train driver dispatcher, who manually adjusts TDR–DSS. If disruption events make the current train driver schedule infeasible, the

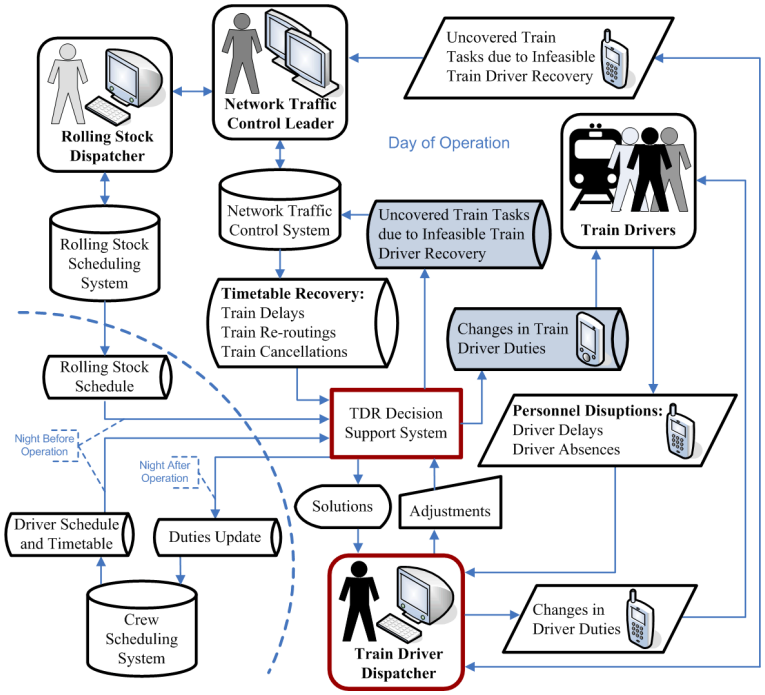


Figure 3.5: Train driver recovery decision support system information flow.

system solves the train driver recovery problem, e.g. with the solution method presented in this thesis or otherwise. The train driver dispatcher chooses one of the suggested solutions if more than one solution is generated and makes adjustments to the solution if necessary. Changes to the train driver duties are reported to the train drivers. If the train driver schedule cannot be repaired, the system alerts the network traffic control system.

When a daily operation is over at approximately 2:30 a.m. the following morning, all changes to the driver duties registered during the operational period are transferred to the crew scheduling system. PDS has an interface to the personnel database (SAP), where overtime work and absences of the train drivers are registered.

3.3.3 Visual Representation of the Prototype

During this project it has become apparent that implementation of a graphical user interface (GUI) is a necessity for presenting the recovery solutions generated by the developed prototype to the planners and dispatchers at Stog. Even though it was not originally planned, the graphical user interface became the starting point of a better understanding and a fruitful cooperation between the Operations and the Research. The colour scheme and the visual appearance of the GUI resembles that of the short-term scheduler of PDS, which is currently used for dispatching purposes. This way the potential end-users of the system did not have to adjust to the new program appearance. Every train driver duty appears as a line of work on a black background. Train tasks, passengering tasks, taxi drives, meal breaks, stand-by tasks, check-ins and check-outs are shown with certain colours and graphical layouts. As an example, a green line represents a train task and three white stars represent a 30 minutes break. The GUI has not been developed for the purpose of user interaction with the system, and only a few other functions besides the train driver schedule graphical representation are employed.

We present an example in order to illustrate the appearance of the GUI. We consider a disruption neighbourhood within a recovery period of 12:00–13:45. The disruption neighbourhood contains 20 train drivers. Among these, 16 train drivers are included because their duties are possibly affected by disruptions. At least one train task in the affected driver duties within the chosen recovery period is disrupted, i.e. delayed, re-routed or cancelled. Moreover, we include 4 reserve train drivers in the disruption neighbourhood. Figure 3.6 shows a disrupted train driver schedule, and Figure 3.7 shows the recovery solution. Table 3.4 gives explanations to the GUI example. For each train driver we describe the part of the original duty within the disruption neighbourhood, where the first and the last task are the commencing and the terminating tasks, respectively. Notice that the terminating tasks of some train drivers departs after 13:45. The difference in recovery periods of the train drivers is caused by the restriction which does not allow the terminating tasks to start at other station than a train driver depot, as explained in Section 3.2.1. Disruptions and their effect on the train driver duties are described in the subsequent columns of the table. The last column in Table

3.4 presents the recovery duties found by solving the TDRP.

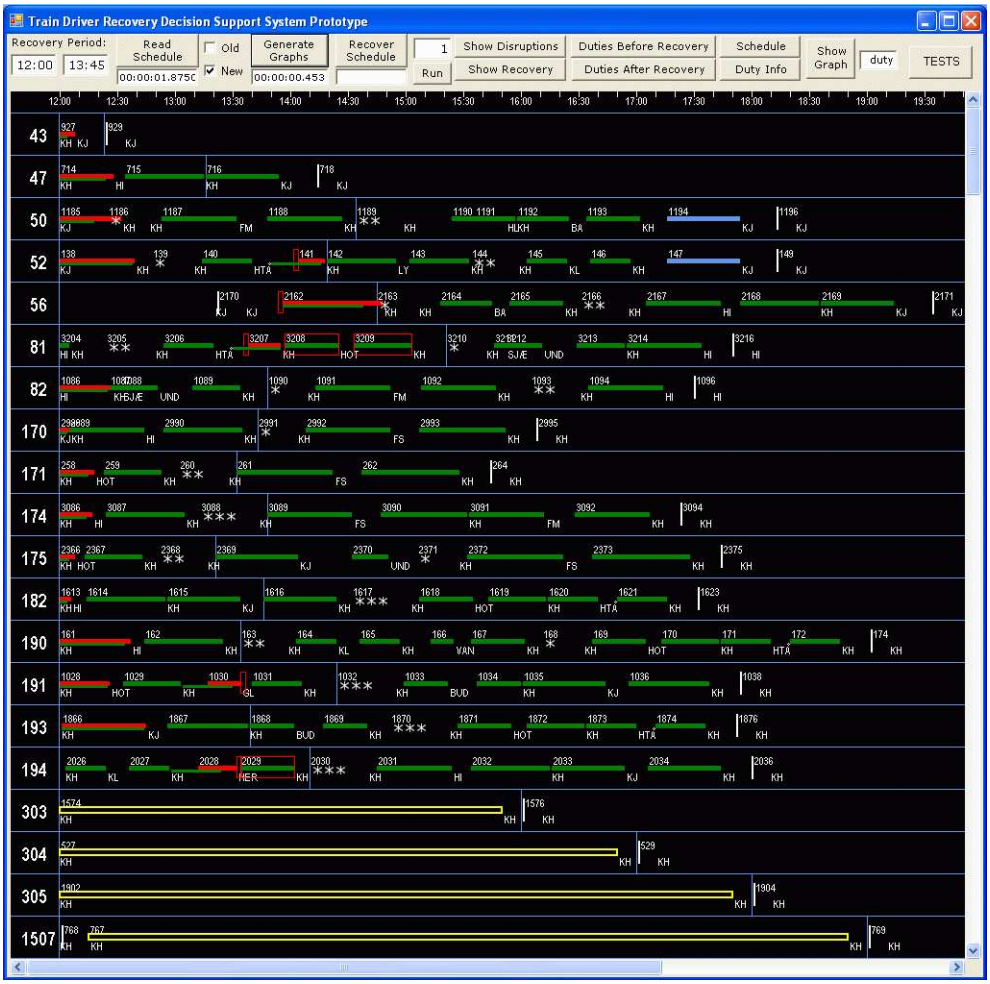


Figure 3.6: Graphical user interface: disrupted schedule.

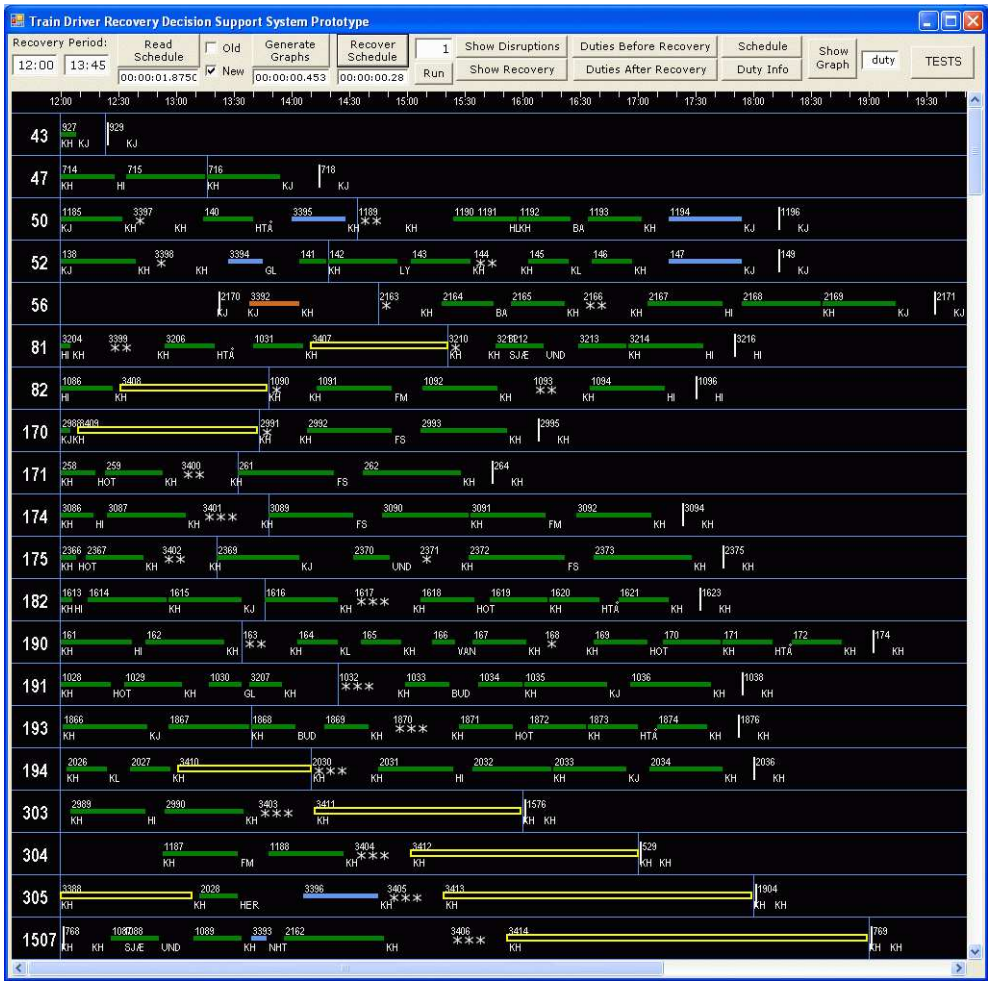


Figure 3.7: Graphical user interface: recovered schedule.

Table 3.4: Description of disruption neighbourhood and solutions in the GUI example.

Nr	Original Duty	Disruption	Disrupt. Effect	Recovery Duty
43	927-929	arr. delay 927 at KJ	none	927-929
47	714-715-716	arr. delay 714 at HI	none	714-715-716
50	1185-PAD-118-PAD	arr. delay 1185 at KH	broken connection 1185-PAD	1185-PAD-140-PAS-PAD
52	138-PAD-140-141-142	arr. delay 138 at KH; re-routing dep. 141 from HTÅ to GL	broken connection 140-141	138-PAD-PAS-141-142
56	CIN-2162-PAD	re-routing dep. 2162 from KJ to NHT; arr. delay 2162 at KH	broken connection CIN-2062-PAD	CIN-TAXI-PAD
81	3204-PAD-3206-3207-3208-3209-PAD	re-routing dep. 3207 from HTÅ to GL; cancel. 3208 from KH to HOT; cancel. 3209 from HOT to KH	broken connection 3206-3207-PAD	3204-PAD-3206-1031-RDG-PAD
82	1086-1087-1088-1089-PAD	arr. delay 1086 at KH	broken connection 1086-1087	1086-RDG-PAD
170	2988-2989-2990-PAD	arr. delay 2988 at KH	broken connection 2988-2989	2988-RDG-PAD
171	258-259-PAD-261	arr. delay 258 at HOT	none	258-259-PAD-261
174	3086-3087-PAU-3089	arr. delay 3086 at HI	none	3086-3087-PAU-3089
175	2366-2367-PAD-2369	arr. delay 2366 at HOT	none	2366-2367-PAD-2369
182	1613-1614-1615-1616	arr. delay 1613 at HI	none	1613-1614-1615-1616
190	161-162-PAD	arr. delay 161 at HI	none	161-162-PAD
191	1028-1029-1030-1031-PAU	arr. delay 1028 at HOT; dep. delay 1030 from KH; re-routing arr. 1030 from HTÅ to GL	broken connection 1028-1029;1030-1031	1028-1029-1030-1031-PAU
193	1866-1867-1868	arr. delay 1866 at KJ	none	1866-1867-1868
194	2026-2027-2028-2029-PAU	dep. delay 2028 from KH; re-routing arr. 2028 from BA to HER; cancel. 2029 from BA to KH	broken connection 2028-PAU	2026-2027-RDG-PAU
303	RDG	none	none	RDG-2989-2990-PAU-RDG
304	RDG	none	none	RDG-1187-1188-PAU-RDG
305	RDG	none	none	RDG-2028-PAS-PAU-RDG
1507	CIN-RDG	none	none	CIN-1087-1088-1089-PAS-2162-PAU-RDG

3.3.4 Implementation Remarks

During irregular operations the train driver dispatcher is often in possession of information which is not yet available in the system or which is difficult to express through e.g. costs of recovery duties in order to influence the recovery solution. It is therefore very important that the train driver recovery decision support system is semi-automatic in a way that it allows human interference with the solution to TDRP.

The end-users of TDR–DSS must be able to e.g. fix {driver, train} and {train i, train j} pairs manually during the generation of recovery duties, i.e. to assign particular train tasks to particular drivers and force a particular train task j to be assigned to the same duty immediately after the train task i. Fixing a {driver, train} pair, i.e. forcing a particular train driver to cover a particular train task is already implemented in the solution method (see Section 5.8.2). Fixing a {train i, train j} pair can be implemented by merging vertices corresponding to train i and train j in duty graphs. Duty graph generation is described in Section 4.2.2. Merging vertices can be implemented by removing all outgoing arcs from the vertex corresponding to train i except from the arc to train j, and at the same time removing all incoming arcs to train j except from the arc from train i. The train driver dispatcher must also be able to generate and adjust disruption neighbourhoods, i.e. to add and remove train drivers and train tasks manually from TDRP problem instances.

Several recovery objectives can be considered to be implemented as standard features of the system, so that a train driver dispatcher could be able to choose a recovery objective according to the particular disrupted situation on the S-train network. At present the objectives of recovery are expressed through the cost of arcs in duty graphs, as described in Section 4.2.4. As an example, by setting higher costs to passengering tasks and deadheading in a taxi, the recovery objective is aimed at minimizing the physical cost of duties, i.e. the cost of taxi rides and the costs of having train drivers engaged in activities which do not involve driving a train. As another example, the recovery objective is aimed at the robustness of the solution by setting high costs on tight connections between train tasks.

We suggest that the TDR–DSS must be developed in tight cooperation with

the end-users, i.e. the train driver dispatchers at S-tog. There are at least two reasons for that. Firstly, constant communication with the end-users is essential for the quality and the speed of the system development and implementation. The train driver dispatchers have operational knowledge, which is incomparable to the knowledge about S-tog operations any software developer can obtain in a short period of time. Secondly, if the end-users are consulted with respect to e.g. the visual appearance and some program features, there will exist an initial acceptance of the system. Introduction of new information technology systems always requires time and efforts from the end-users for learning and growing accustomed to the new system. This transition time will be shortened if many end-users are initially involved in the system development.

Chapter 4

Model and Network

A set partitioning problem formulation of the train driver recovery problem is presented in this chapter. We show that the linear programming (LP) relaxation of the problem formulation has strong integer properties due to the structure of the constraint matrix. This explains why solutions to the LP relaxation of the problem are integral in the majority of test cases.

The construction of the network used for recovery duty generation is also presented in this chapter. We show that a resource constrained path from the source vertex to the sink vertex of the generated network satisfies all requirements for recovery duty feasibility.

4.1 Integer Programming Model

4.1.1 Set Partitioning Problem Formulation

We formulate the train driver recovery problem (TDRP) as a set partitioning problem. Let K be the set of train drivers and N be the set of train tasks that need to be covered within the chosen disruption neighbourhood. Let R^k be the set of feasible recovery duties for a driver $k \in K$. Recovery duty feasibility is discussed in Section 4.2.1.

The cost c_r^k reflects the unattractiveness of the recovery duty $r \in R^k$ for the driver $k \in K$ in the recovery schedule. A binary decision variable x_r^k equals 1 if the duty $r \in R^k$ for the driver $k \in K$ is included in the recovery solution and equals 0 otherwise. A binary parameter a_{ir}^k is used to define whether or not the task $i \in N$ is covered by the duty $r \in R^k$.

$$\text{(TDRP)} \qquad \text{Minimize} \quad \sum_{k \in K} \sum_{r \in R^k} c_r^k x_r^k \tag{4.1}$$

$$\text{Subject to} \quad \sum_{r \in R^k} x_r^k = 1 \qquad \forall k \in K, \tag{4.2}$$

$$\sum_{k \in K} \sum_{r \in R^k} a_{ir}^k x_r^k = 1 \qquad \forall i \in N, \tag{4.3}$$

$$x_r^k \in \{1, 0\} \quad \forall r \in R^k, \ \forall k \in K. \tag{4.4}$$

The train driver recovery problem (4.1) – (4.4) aims at finding a minimum cost set of feasible train driver recovery duties for train drivers within the disruption neighbourhood, such that all train tasks within the recovery period are covered. Even though the TDRP can be viewed as a feasibility problem, it is important to reflect the stability (or robustness) of the recovery solution by minimizing the number of modifications from the original schedule. This can be achieved by using a disruption neighbourhood which is as small as possible and by setting high costs in the objective function (4.1) to highly modified recovery duties.

The *train driver constraints* (4.2) ensure that each train driver is assigned to exactly one recovery duty in the recovery schedule. The train driver constraints have a generalised upper bound (GUB) structure, since the constraints are disjoint and each column contributes to exactly one driver constraint. The *train task constraints* (4.3) have a set partitioning structure and ensure that each train task in the recovery schedule is covered exactly once. Constraints (4.4) are the integer constraints of the model. Since the model represents a recovery problem, all "global" constraints, which are usually relevant for a train driver scheduling problem, are not relevant here. For example, the maximum number of long duties in the schedule or the number of drivers starting at a certain depot is not important for the schedule recovery.

The TDRP model (4.1)–(4.4) has the pure set partitioning problem structure of a crew rostering problem. The generalised set partitioning formulation of the crew rostering problem is used in real-life applications within the airline industry for e.g. Air New Zealand Butchers et al. [2001] and Air France Gamache et al. [1999]. We should draw attention to particular properties of the TDRP model. First, the column structure of the matrix specified by (4.3) satisfies many implicit constraints which govern the feasibility of recovery duties. These constraints are therefore not explicitly added to the model. Second, as observed in Ryan and Falkner [1988], the linear programming relaxation of the set partitioning formulation of the crew rostering problem possesses strong integer properties due to the existence of the generalised upper bound crew constraints, which give the perfect structure of the submatrix, corresponding to each crew member. The structure of the constraint matrix A of the train driver recovery problem has a structure similar to the set partitioning formulation of the crew rostering problem. It implies that the linear programming relaxation of the train driver recovery problem also possesses strong integer properties. We discuss this relationship to perfect matrix theory (Padberg [1974]) in the next section.

4.1.2 Integer Properties of the Model

Four classes of matrices are known to have the *integral property*: *totally unimodular* matrices (Hoffman and Kruskal [1956]), *balanced* matrices (Berge [1972]), *perfect* matrices (Padberg [1974]) and *ideal* matrices (Lehman [1979],

Padberg [1993]). A survey of some results on perfect, ideal and balanced matrices is presented by Conforti et al. [2001]. Having a set partitioning problem with the constraint matrix that has the integral property implies that the feasible region of the LP-relaxation of the problem has extreme points that are all integral. Having a feasible region with only integral extreme points means that one only needs to solve the LP-relaxation to obtain the optimal integer solution to a set partitioning problem. Note that when multiple optimal solutions exist, fractional solutions with an equivalent objective function value also exist. Ryan and Falkner [1988] describe characteristics of scheduling set partitioning models which can be used in order to exploit the unimodular, balanced and perfect structure of the constraint matrices.

The constraint matrix of the train driver recovery problem has a block structure, where every block of columns corresponding to recovery duties for one train driver is represented by a perfect matrix. This is demonstrated in the following example. Let A be a zero–one matrix corresponding to the constraints (4.2)–(4.3) of the TDRP. A is an $m \times n$ matrix, where $m = |K| + |N|$ is the number of rows in the problem and $n = \sum_{k \in K} |R^k|$ is the number of columns. Let A^k be a submatrix of A corresponding to columns belonging to the driver k. A^k is an $m \times n^k$ matrix, where $n^k = |R^k|$ is the number of columns in the problem belonging to the same driver k and m is the number of rows in A. An example of a matrix A for a disruption neighbourhood involving $|K| = 4$ drivers who are to be assigned to $|N| = 3$ train tasks is shown on Figure 4.1.

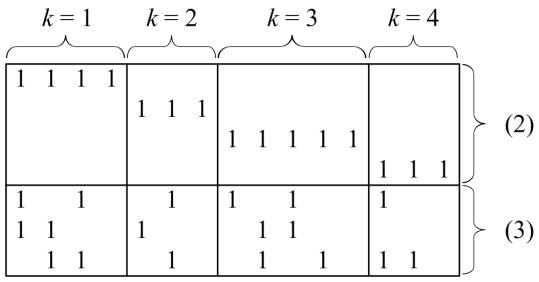


Figure 4.1: Example of a constraint matrix structure of the TDRP.

According to Theorem 3.16 by Padberg [1974], an $m \times n$ matrix A is perfect if and only if it does not contain any $m \times l$ submatrices A_l, where $3 \leq l \leq n$,

with the following property denoted $\pi_{\beta,l}$: A_l contains an $l \times l$ nonsingular submatrix B_l with row and column sums all equal to $\beta \geq 2$, while each row of A_l, which is not in B_l, is either componentwise equal to a row in B_l or has a row sum strictly less than β. Padberg refers to submatrices with such a property as to "forbidden submatrices".

In order to illustrate the theorem, let us examine the 4×4 matrix A^1 on Figure 4.2. A^1 is the submatrix of A from Figure 4.1, which belongs to the driver $k = 1$. Rows with only zero entries are neglected. All $m \times l$ submatrices of A^1 for $3 \leq l \leq 4$ are shown in Figure 4.3. According to the theorem, in order for A^1 to be perfect, the five matrices shown in Figure 4.3 must not contain any $l \times l$ nonsingular submatrices with the $\pi_{\beta,l}$ property, where $\beta \geq 2$ and $3 \leq l \leq 4$. As an example, let us generate all nonsingular $l \times l$ submatrices B_l for $l = 3$ from the 4×3 submatrix A^1_3. The four generated 3×3 submatrices are shown on Figure 4.4. All matrices are nonsingular.

The only one matrix among the four B_3 matrices with equal sums of rows and columns is B^3_3 with $\beta = 2$. The only row of A^1_3, which is not in B^3_3, is row 1, which is the train driver constraint represented by a vector $(1, 1, 1)$. The row sum of this row is $3 > \beta$. Hence, A^1_3 does not contain any submatrices with the $\pi_{\beta,l}$ property. The procedure of generating $l \times l$ nonsingular matrices with $l = 3, 4$ and equal sums of rows and columns is applied to A^2_3, A^3_3 and A^1_4. None of the matrices contains any submatrices with the $\pi_{\beta,l}$ property. Hence, the matrix A^1 is perfect. According to Padberg [1974], an $m \times n$ zero–one matrix A is *perfect* if the polytope $P = \{x \in \mathbb{R}^n : Ax \leq \tilde{e}, x_j \geq 0, j = 1, ..., n\}$, where $\tilde{e}^T = (1, ..., 1)$ with m components, has only integral vertices, i.e. if $P = P_I = \text{conv}\{x \in P : x_j = 0 \text{ or } 1, j = 1, ..., n\}$.

	1	2	3	4
1	1	1	1	1
2	1		1	
3	1	1		
4		1	1	

A^1

Figure 4.2: Driver $k = 1$ submatrix A^1 of A.

Every submatrix A^k of A has a row corresponding to the train driver GUB

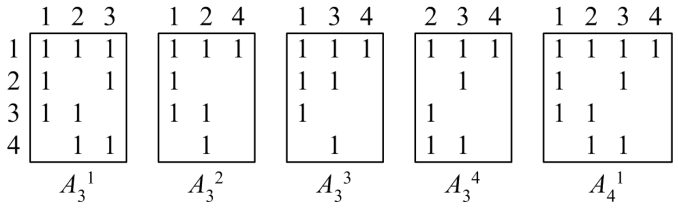


Figure 4.3: All $m \times l$ submatrices of A^1 with $l = 3, 4$.

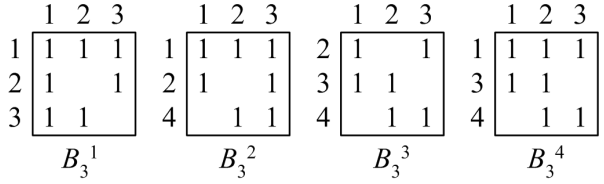


Figure 4.4: All $l \times l$ submatrices of A_3^1 with $l = 3$.

constraint (4.2). As observed by Ryan and Falkner [1988], the GUB crew constraint in the SPP formulation of the crew rostering problem is a "dominant" row in the submatrix of the crew member k, which prevents the existence of the $\pi_{\beta,l}$ property and ensures that each A^k is perfect. Indeed, none of the $m \times l$ submatrices A_l where $3 \leq l \leq n$ can contain a nonsingular submatrix B_l with the row sum strictly larger than the dominant row corresponding to (4.2). Therefore, due to the existence of the dominant train driver constraint, every driver submatrix A^k of A is perfect according to Padberg [1974].

The perfectness of a submatrix A^k or any other matrix with a GUB constraint can also be shown using graph theory. Let G_I^k be the *intersection graph* associated with a driver submatrix A^k. The nodes of the intersection graph G_I^k correspond to columns of A^k and every pair of nodes is connected by an edge in G_I^k if and only if the two corresponding columns in A^k have a common 1-entry in some row. An example of the intersection graph G_I^1 associated with the driver matrix A^1 of driver $k = 1$ is shown in Figure 4.5. Note that graph G_I^1 is complete.

A graph is *perfect* if for every induced subgraph, the chromatic number is

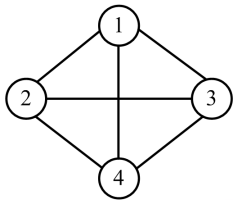


Figure 4.5: Intersection graph G_I^1 associated with A^1.

equal to the cardinality of the maximal complete subgraph (i.e. the *maximal clique*) (see Lovász [1972], re-printed in Lovász [2006]). The *chromatic number* is the number of different colours necessary to colour all nodes in the graph such that adjacent nodes have different colours. Every subgraph induced from any intersection graph G_I^k of the driver submatrix A^k of TDRP is complete, since the GUB constraint ensures that all nodes in the graph are connected. The chromatic number of a complete graph equals the number of nodes in the graph. A maximal complete subgraph (i.e. a maximal clique) of a complete graph is the graph itself with the cardinality equal to the number of nodes in the graph. Hence, in any induced subgraph of an intersection graph G_I^k of any driver submatrix the chromatic number equals to the maximum size of the complete subgraph. Therefore, G_I^k is perfect for any choice of k.

According to Padberg [1974], a necessary condition for a zero–one matrix to be perfect is that it contains the incidence vectors of all maximal cliques of the associated intersection graph. A zero–one matrix, whose columns are indexed by the nodes of the associated intersection graph G_I and for which every row is the incidence vector of a maximal clique of G_I is called a *clique-matrix*. Padberg [1974] states that a clique-matrix is perfect if and only if the associated intersection graph is perfect. As an example, the clique-matrix of G_I^1 is the GUB row of matrix A^1, i.e. the train driver constraint (4.2) of the driver $k = 1$. Since G_I^k is perfect for any $k \in K$, then the clique-matrix of any G_I^k is perfect. Since each matrix A^k contains the incidence vector of the maximal clique of G_I^k, i.e. the GUB row, any driver submatrix A^k is perfect. All other rows in A^k are not essential for explaining why the matrix is perfect, as long as A^k contains a GUB row.

Since every submatrix representing one driver's block of columns is perfect, fractional solutions will never appear within one driver's block of recovery duties. Any fractions in an optimal solution to the LP-relaxation of the TDRP can therefore only occur between blocks of columns belonging to different drivers. In other words, if a fractional solution occurs, it means that two or more drivers compete for one or more train tasks in their recovery duties. Two drivers can only compete for the same train task, if both drivers are available at the departure station prior the time of the train departure. As mentioned in Section 1.4.1, in the standard S-tog's train driver schedule, the train tasks arriving at terminal stations have unique subsequent tasks due to the geographical position of the stations and the train line pattern in the S-train network. Hence the number of train tasks each driver can compete for with other drivers in the planned S-tog schedule is very limited. The situation can be different in recovery situations, where train driver duties can be constructed in a less restrictive way, and "unnecessary" deadheading tasks, which are usually not allowed in the planned schedules, can contribute to more competition for particular train tasks among the drivers. However, since only a limited number of train drivers and train tasks are included in the disruption neighbourhood, even for recovery problems there is a high probability that optimal solutions to the LP-relaxations of many problems are integral. This is confirmed by the computational experiences described in Chapter 6.

4.2 Modelling Recovery Duties

The TDRP model (4.1)–(4.4) is characterized by a potentially large number of columns, corresponding to the number of feasible recovery duties for all drivers within the disruption neighbourhood, even for a relatively small number of rows given by the number of train drivers and the number of train tasks within the disruption neighbourhood. Efficient methods for recovery duty generation are therefore an important issue in the solution process. Duties for the crew scheduling and re-scheduling are often generated from the *time-line* networks, *connection* networks, *time-band* networks (see Clausen et al. [2009] for an overview) or *duty-period-based* networks (Nissen and Haase [2006]). It is important to choose an efficient network structure for TDRP,

since the network is built for every disrupted situation and updated when the disruption neighbourhood is expanded. There is a computational trade-off between the network generation and the duty generation. The more information expressed through the network, the faster the duties can be generated, since fewer constraints have to be taken care of in the duty generation process. During this project, the network representation as well as the overall data structure has been constantly modified and updated in order to adjust to the new information about changes in the duty rules and to accommodate different timetables, train driver schedule representations and test scenarios.

4.2.1 Recovery Duty Feasibility Conditions

A recovery duty is feasible, if the three following conditions are satisfied:

Condition 1: Task subsequences feasibility

At the completion of every train task or other activity in a feasible recovery duty of a train driver we can identify a set of activities, which can subsequently be assigned to the driver. These activities are either alternative train tasks, meal breaks, passengering tasks or check-out tasks. We refer to a particular subsequent activity in a train driver duty as a *subsequence* of a task. Planners at S-tog use an abbreviation for every activity in the train driver schedule. These are presented in Table 4.1.

A sequence of two activities (i, j) must satisfy following constraints: the arrival station of i must coincide with the departure station of j and the start/departure time of j must occur later than the finishing/arrival time of i plus a certain minimum technical connection time. Technical connection times components are presented in Table 1.1. Minimum technical connection times used in this project are presented in Table 4.2. Note that a passengering task (PAS) from i to j requires that there is a train departing from the arrival station of i, which stops at the departure station of j.

Table 4.1: Abbreviations of activities in the train driver schedule at S-tog.

Abbr.	Task Description
TFF	Train task.
PAS	Passengering task on a train.
TAXI	Deadheading in a taxi.
RDG	Stand-by activity.
PAU	Long meal break (30 minutes).
PAD	Short meal break (20–25 minutes).
CIN	Check-in activity (15 minutes).
CUD	Check-out activity (10 minutes).

Table 4.2: Minimum connection times between subsequent tasks in duties.

Activity i	Minimum Connection Time	Activity j
PAD/PAU/CIN	BEV+PIT	TFF
TFF	PIF+BEV	PAD/PAU/CUD
CIN	BEV	PAS
PAS	BEV	CUD
TFF	SPD	PAS
PAS	SPD	TFF
TFF	SPD	TFF
PAS	SPD	PAS

Condition 2: Recovery duty length feasibility

A recovery duty must not begin before the driver is available after finishing the last task he/she is performing according to the original schedule prior the disruption. If the original duty finishes within the recovery period, the recovery duty must not exceed the original duty length unless a duty length extension is explicitly allowed. For this project it is only allowed to shift the start time of the check-out activity in a recovery duty, when a delayed train in the disruption neighbourhood is the only possibility for the driver to be transferred to the crew depot station for the check-out (either by being assigned to the train task or as a passenger on the train). If the original duty finishes after the end of the recovery period, the driver is required to be present to cover his/her original duty after the recovery end time. Therefore,

the duty length is identified by both the times and the stations, at which the train driver begins the recovery duty and where the driver must be available at the end of his recovery duty.

It should be mentioned here that the view of recovery adopted in this thesis is strict, requiring all drivers to be back to their original duties at the end of the recovery period. If a driver cannot get back to the original duty at the end of the recovery period, the recovery period for that driver is extended by a certain time period until the feasibility of the duty is established (see Section 3.2.2). An alternative view of recovery is to relax the requirement that drivers be back to their original duty at the end of a recovery duty. As long as duty feasibility is achieved at the beginning of the recovery period, dispatch decisions may be implemented immediately leaving the infeasibilities at the ends of recovery duties to be resolved by subsequent optimizations as the recovery process proceeds.

Condition 3: Meal break feasibility

The recovery period is very likely to cover one or two originally scheduled meal breaks for drivers in K. According to the trade union agreement, drivers are entitled to a meal break after driving a train for at most 3. However, in most of the planned duties, the driving time between breaks is less than 3 hours. The train drivers usually prefer two half-breaks (at least 45 minutes in total) in the duty instead of one full break (at least 30 minutes in total). It is therefore considered a reduction in the duty quality, if in a disrupted situation a three-block duty is transformed into a two-block duty.

In order to account for meal break requirements and keep the duty quality as high as possible, a feasible recovery duty must satisfy the following: all originally scheduled driver's breaks within the recovery period are held, each break starts within a certain time interval, e.g. +/- 10 minutes, of the originally scheduled break start, and the type of break corresponds to that originally scheduled (either a full break or one of the two half-breaks without violating the total break time). It is feasible to hold longer breaks, but as mentioned previously any time in excess of the required break duration is considered to be a buffer time between two train tasks. The break can be held at either of the two crew depots, no matter which crew depot the driver is originally assigned to.

4.2.2 Duty Graph Generation

A directed acyclic *duty graph* $G^k = (V^k, A^k)$ is constructed for every driver $k \in K$, where V^k is the set of vertices and A^k is the set of arcs in the graph. V^k contains a source vertex o^k, a sink vertex d^k, and a set of train task vertices N^k within the disruption neighbourhood. The source vertex $o^k \in V^k$ represents the commencing task in the duty of driver k, while the sink vertex $d^k \in V^k$ represents the corresponding terminating task. Figure 4.6 shows the train tasks in the expanded disruption neighbourhood presented in Figure 3.2 on page 61.

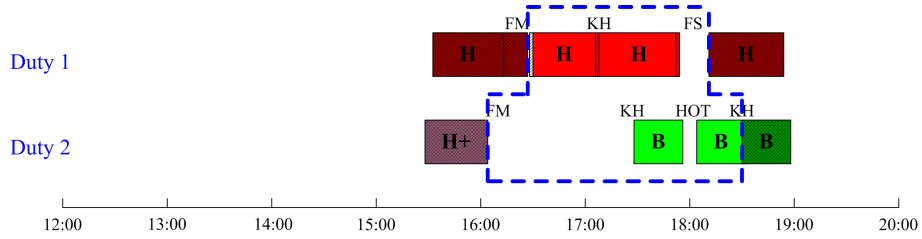


Figure 4.6: Train tasks in disruption neighbourhood example.

The set of arcs A^k represents feasible connections between tasks in V^k. A task w is a feasible subsequence of a task v if the departure time of w is later than or equal to the ready time $t_{\text{ready}}(v, w)$ and if the departure station of w coincides either with the arrival station of v or the arrival station of the deadheading task assigned to the driver from the arrival station of v to the departure station of w. The *ready time* $t_{\text{ready}}(v, w)$ is calculated as the arrival time of v plus a certain time span, which is necessary for the driver to conduct intermediate actions after finishing the task v in order to begin the task w. The actions might only include the minimum technical connection times between two activities, as described in Section 4.2.1, or also include other activities, such as meal breaks or passengering tasks or both.

As an example shown in Figure 4.7, if the driver holds a full break between the train task v and the train task w and the arrival station of v is the same as the departure station of w, a time for handing the train over to another driver (PIF), walking from the platform to the crew depot (BEV), holding

the long break (PAU), walking from the crew depot back to the platform (BEV) and taking over the train from another driver (PIT) is added to the arrival time of the train task v in order to calculate $t_{\text{ready}}(v, w)$.

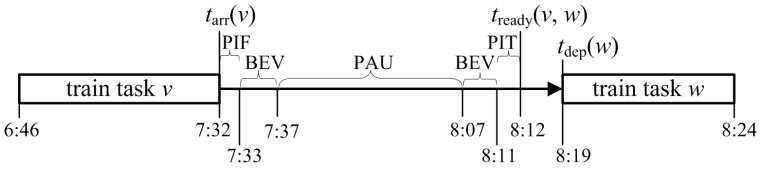


Figure 4.7: Feasible subsequence with a long break.

The example in Figure 4.8 shows the ready time $t_{\text{ready}}(v, w)$ of the check-in task v, where the subsequent train task w departs from a different station than the check-in depot station of v, and w is the earliest morning task of the particular train unit. The driver needs to walk from the depot to the platform of the passengering train task (BEV), ride as a passenger on a train to the departure station of w (PAS), walk to the departure platform (BEV) and make the train ready for driving (KLG). Since platforms at some stations are situated further away from the depot and other platforms, the minimum required duration of BEV is station-dependent. Durations of all technical connection times in the S-tog train driver schedule are shown in Table 1.1 in Section 1.4.1.

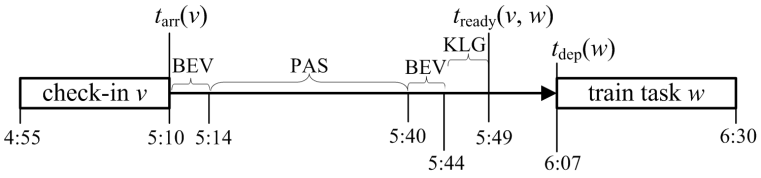


Figure 4.8: Feasible subsequence with a passengering task.

Three general arc types can be identified: *break opportunity* arcs, where at least one break type (a full break or a half-break) can be held between the train tasks, *deadheading* arcs, where either a passengering task or a taxi ride is required for covering the sequence of tasks represented by the two connected vertices and *train connection* arcs, including all other connections

from and/or to train tasks, where meal breaks are not possible and deadheadings are not required.

A graph generation process can be time consuming. Particulary, passengering arcs are timewise costly to generate, since it is necessary to identify if there are other trains available for deadheading the driver from one station to another. At most two train tasks are searched for when passengering arcs are generated. This increases the chance of finding a passengering possibility from e.g. a terminal station of one train line to a terminal station of another train line. In a real-life disrupted situation, passengering of a driver can take place from one intermediate station to another, but this possibility is not often utilised by the train driver dispatchers due to the complexity of the duty swap combinations.

Some arcs in G^k are identical for several choices of train driver $k \in K$, since more than one train driver is usually able to cover the same sequence of train tasks within the recovery period. In order to reduce the computational time for the graph generation at every instance of TDRP, we generate a *common driver graph* $G = (V, A)$, where the set of vertices V contains all source vertices $o^k, \forall k \in K$, all sink vertices $d^k, \forall k \in K$ and all train tasks N in the disruption neighbourhood. As an example, a common duty graph G, shown on Figure 4.9, is generated for the disruption neighbourhood shown on Figure 4.6. Table 4.3 provides the description of the arc types with the corresponding ready times, which are used for generating G.

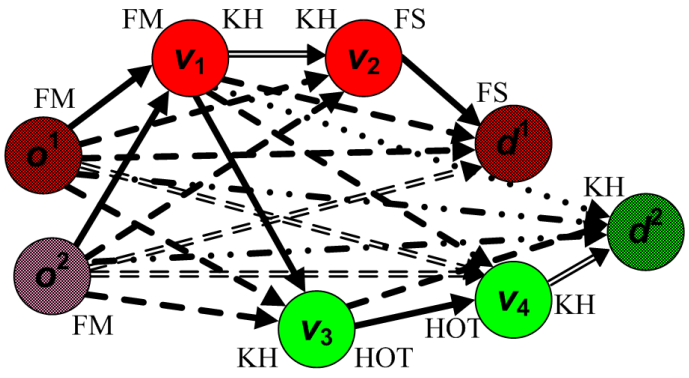
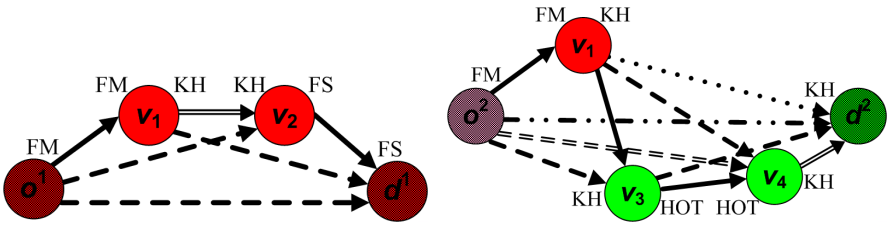


Figure 4.9: Example of a duty graph G.

After the common driver graph G is generated, duty graphs $G^k, \forall k \in K$ are induced from G by copying a part of the data structure from G. For every train driver k we identify the set of train tasks N^k. Vertex $v \in N$ is only included in the set N^k of the duty graph G^k, if the corresponding train task can potentially be covered by the driver k within his or her recovery duty without considering the meal break requirements of the duty. The adjacency list data structure of the graph allows transferring the outgoing directed arcs directly from A to A^k, when the set of vertices N^k is collected. The process of inducing G^k from G for every k is computationally more efficient than generating each G^k due to the computationally expensive arc generation process. Duty graphs G^1 and G^2 induced from the common graph G shown on Figure 4.9 are illustrated in Figure 4.10. It should be mentioned that generation of "personal" duty graphs G^k corresponds to preprocessing techniques similar to the one presented in Aneja et al. [1983] and Dumitrescu and Boland [2001] in the context of the label setting algorithm for the resource constrained shortest path on an acyclic graph, which is used for recovery duty generation (see Section 5.4.3).



(a) Duty graph $G^1 \subset G$.

(b) Duty graph $G^2 \subset G$.

Figure 4.10: Duty graphs induced from G.

Table 4.3: Arc types used in duty graph examples.

Arc Type	Arc Description	Ready Time $t_{\text{ready}}(v, w)$
⟹	Connects two train tasks of the same line following the same direction.	$t_{\text{arr}}(v)$
⟶	Connects two train tasks, where a platform walk is required.	$t_{\text{arr}}(v)$ + SPD
⟶	A passengering arc; requires the existence of one train task on which a passengering task can be performed.	$t_{\text{arr}}(v)$ + SPD + PAS(v, w) + SPD
⟶	A double passengering arc; requires the existence of two train tasks on which passengering tasks to and from the central station København H can be performed.	$t_{\text{arr}}(v)$ + SPD + PAS(v,KH) + SPD + PAS(KH,w) + SPD
⟶	A break opportunity arc; requires that the arrival station of v is a crew depot.	$t_{\text{arr}}(v)$ + PIF + BEV(v) + 20 min + BEV(v) + PIT
⟶	A combined passengering and break opportunity arc.	$t_{\text{arr}}(v)$ + SPD + PAS(v, w) + BEV(w) + 20 min + BEV(w) + PIT

4.2.3 Feasible Recovery Duty on a Graph

Any directed path from the source vertex o^k to the sink vertex d^k in the duty graph G^k represents a sequence of vertices, starting at the driver k's commencing task in the duty and ending at the terminating task after the end of the recovery period. Any directed path from o^k to d^k satisfies the first two recovery duty feasibility conditions described in Section 4.2.1. Condition 1, the task subsequences feasibility, is satisfied since arcs in the duty graphs represent feasible connections between tasks. Condition 2, the recovery duty length feasibility, is satisfied implicitly by the source vertex o^k and the sink

vertex d^k for each driver k in G.

A *resource constrained directed path* p from o^k to d^k in G^k represents a *feasible recovery duty* $r \in R^k$, if the path satisfies the following resource constraint: for every originally scheduled meal break within the recovery period the path must contain at least one arc with a break opportunity, which begins within a certain time interval from the originally scheduled start of the break (e.g. +/- 20 minutes), and for which the total length of the break opportunity arcs in the path corresponds to at least the length of the total break time originally scheduled within the recovery period. Such a resource constrained path also satisfies Condition 3, the meal break feasibility of the recovery duty. As an example, Figure 4.11 shows a resource constrained path on the duty graph G^2, which represents a feasible recovery duty for the driver assigned to Duty 2.

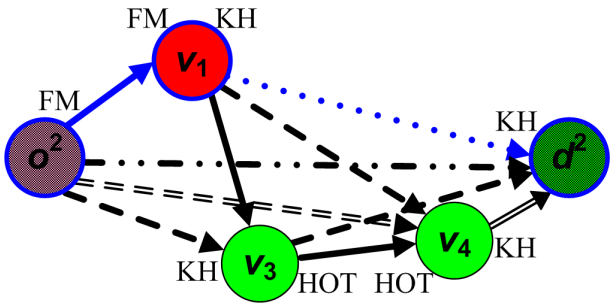


Figure 4.11: A feasible recovery duty path $o^2 \rightarrow v_1 \rightarrow d^2$ in G^2.

The cost $c(p)$ of the resource constrained path $p = o^k \rightarrow v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_h \rightarrow d^k$ in G^k is a sum of *arc costs* and *vertex costs*. Let $c(v, w)$ be the cost of arc (v, w) and $c(v)$ be the cost of vertex v. Then the cost of path is expressed in equation (4.5), where h is the number of intermediate train task vertices on the path p between the source o^k and the sink d^k.

$$c(p) = [c(o^k, v_1) + \sum_{i=1}^{h-1} c(v_i, v_{i+1}) + c(v_h, d^k)] + [c(o^k) + \sum_{i=1}^{h} c(v_i) + c(d^k)]. \quad (4.5)$$

4.2.4 Recovery Duty Cost

In practice it is very difficult to determine the actual cost of a recovery duty $r \in R^k$ for a train driver $k \in K$. In order to keep the recovery duty cost calculations simple and fast, the *recovery duty cost* c_r^k in the set partitioning formulation of TDRP is linear and is represented by the sum of costs of activity sequences assigned to the duty, i.e. by the *sum of arc costs* of the path $p = o^k \rightarrow v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_h \rightarrow d^k$, which represents the recovery duty r:

$$c_r^k = c(o^k, v_1) + \sum_{i=1}^{h-1} c(v_i, v_{i+1}) + c(v_h, d^k). \tag{4.6}$$

The arc cost $c(v, w)$ of an arc (v, w) in G^k reflects the unattractiveness of the connection from v to w for the train driver $k \in K$. The objective of the recovery can be expressed by setting higher costs on arcs which contradict with the objective. In order to minimize the number of modifications in the recovery duties from the original schedule, a sequence of tasks is assigned a zero cost if it appears in the original duty of the driver k, i.e., if both activities v and w belong to the original driver duty. Slightly less preferable is an arc (v, w), where the train task v is *not* in the original duty of the driver, but the task w is originally scheduled in the duty of the driver. Such arcs are preferable since there is a chance that the driver continues with his original duty after completion of the task w. These arcs are assigned only a little cost.

Deadheading is in general not preferable unless no other option is available, since the deadheading train driver is not engaged in a working task. Unscheduled deadheading in a taxi is more costly than passengering on another train, since the taxi expenses have to be covered.

The difference between the departure time $t_{\text{dep}}(w)$ and the ready time $t_{\text{ready}}(v, w)$ of the arc (v, w) is the *buffer time* or the *idle time* $t_{\text{idle}}(v, w)$, dependent on how long the time period is. A feasible subsequence without any buffer time is very tight and non-robust with respect to even small delays. On the other hand, a very long idle time is not attractive either, particularly at terminal stations without crew facilities. In order to control the tightness of the

schedule and the unwanted idle time, a minimum required buffer time $\tau^{\min}_{\text{buffer}}$ and a maximum allowed idle time $\tau^{\max}_{\text{idle}}$ are defined and used when setting the cost of the arcs. Furthermore, we also define a *minimum reserve time* $\tau^{\min}_{\text{reserve}}$ as the time which is necessary for being able to cover at least one other train task between $t_{\text{ready}}(v, w)$ and $t_{\text{dep}}(w)$. If the idle time of an arc exceeds $\tau^{\min}_{\text{reserve}}$ and the departure station of w is a crew depot, the train driver can potentially stay in reserve, and such arc must not be penalized as much as connections where a driver can only inactively wait for the next train task.

As a suggestion, the intervals can be defined with $\tau^{\min}_{\text{buffer}} = 2$ min, $\tau^{\max}_{\text{idle}} = 15$ min, and $\tau^{\min}_{\text{reserve}} = 1$ hour. Then connections with the buffer time of less than 2 min are given a high cost in order to promote the robustness of the schedule. Arcs with the buffer time between 2 and 15 min are acceptable for connections during disruptions. Train connections at non-depot stations with the idle time longer than 15 min are given a high cost, while arcs at depot stations with a reserve possibility is not penalized just as much.

Connections without changing the rolling stock are preferred in disruption situations as well as during the schedule planning. Even with the rolling stock change, sequences of train tasks on the same train line are preferable. Whenever possible, train driver changes at København H (KH) must be avoided in order to avoid bottleneck effects due to driver delays. Arcs representing train changes at the central station are therefore given a large cost. Description of the majority of arc types implemented in duty graphs are listed in Table 4.4. All other arcs are combinations of that presented in the table, e.g. a TaxiBreak arc.

The same arc type can be useful for one train driver and unnecessary for another. As an example, break opportunity arcs in G^k are useful if the train driver k is originally assigned a meal break in the duty, and it is not either too late or too early to assign a break at a particular break opportunity arc. In this case the break opportunity arc must be given a little cost. On the other hand, if the original break is scheduled outside the recovery period of a particular driver k', the same break opportunity arc in the duty graph $G^{k'}$ represents an unnecessary idle time. In this case the arc with a break opportunity must be changed to a train change arc with a high cost.

A more sophisticated way of setting costs on recovery duties is preferable in

Table 4.4: Arc descriptions.

Arc type	Description
Original	Represents a sequence of tasks which are present in the original duty of the train driver.
InDuty	Leads to a task which is present in the original driver duty.
InDutyPas	Leads to a task which is present in the original driver duty, requires one or two passengering tasks.
Immediate	Represents a sequence of train tasks with the same train number or a sequence of tasks (v, w), where w represents the first train of the same line which departs from a terminal station immediately after v.
Break	Break opportunity arc. Minimum length of the break opportunity is 20 min.
BreakPas	Break opportunity arc, which requires a passengering task.
Reserve	An arc at a crew depot with $t_{\text{idle}}(v, w) > \tau^{\min}_{\text{reserve}}$.
TrainKH	Train change at KH without break or reserve.
Taxi	Deadheading in a taxi.
Pas	Contents one or two passengering tasks. The arc does not contain break opportunities.
TrainTight	Train change at other stations than KH with $t_{\text{idle}}(v, w) \leq \tau^{\min}_{\text{buffer}}$.
TrainChange	Train change with $\tau^{\min}_{\text{buffer}} < t_{\text{idle}}(v, w) \leq \tau^{\max}_{\text{idle}}$.
TrainIdle	Train change with $\tau^{\max}_{\text{idle}} < t_{\text{idle}}(v, w) \leq \tau^{\min}_{\text{reserve}}$.
Relief	Relief opportunity arc (v, w), where w is a terminating task in G^k, corresponding to a check-out task. With $t_{\text{idle}}(v, w) \leq \tau^{\max}_{\text{idle}}$.
ReliefIdle	Relief arc with $\tau^{\max}_{\text{idle}} < t_{\text{idle}}(v, w) < \tau^{\min}_{\text{reserve}}$.

the train driver recovery decision support system, since the duty cost is not necessarily linear in the cost of train task subsequences within the recovery duty. A more sophisticated cost scheme is the subject of future research. The cost of recovery duty is often a subject to dispatcher's evaluation of the disrupted situation and eventually the specific train driver's wishes and agreements between the train driver and the dispatcher in the disrupted situation. It is therefore important for the recovery solution quality of the decision support that the train driver dispatcher is given an opportunity to manually apply penalties on arc costs of the duty graphs.

Chapter 5

Solution Approach

In this chapter a solution method to the set partitioning based model of the train driver recovery problem is presented. The problem is solved using a branch-and-price framework, where the linear programming (LP) relaxation of TDRP (4.1) – (4.4), abbreviated TDRP–LP, is solved with column generation and dynamic expansion of the disruption neighbourhood. We propose several pricing strategies for the branch-and-price algorithm, a heuristic solution method based on limited subsequences strategy for solving TDRP–LP and a method for expanding the disruption neighbourhood. Integer solutions are found using constraint branching combined with a depth-first search of the branch-and-bound tree.

5.1 Choosing the Solution Method

When solving the train driver recovery problem, only the part of the train driver schedule within the disruption neighbourhood is altered. The parts of the original duties of the involved train drivers outside the recovery period, as well as original duties of train drivers not included in the disruption neighbourhood remain unchanged. This is consistent with the main objective of the recovery problem, which is to minimize the number of modifications from the original schedule. It is only if the solution with the considered disruption neighbourhood is infeasible, e.g. contains one or more non-covered train tasks, that the disruption neighbourhood is expanded.

The recovery duties represented by columns in the set partitioning problem can be generated a priori for all drivers within the disruption neighbourhood. However, the a priori duty generation can be a costly procedure in an operational environment even for a relatively limited number of drivers and train tasks. We therefore implement a framework for a dynamic column generation. Being able to generate columns "on the fly" is particulary efficient in the expansion of the disruption neighbourhood.

Feasible recovery duties for train drivers can be constructed under the restriction that the number of choices for performing different tasks after finishing a task in the duty is limited, even though many possibilities (i.e. many subsequences) might exist. Ryan [1992] describe the use of *limited subsequence filtering* when generating duties for the crew rostering problem of Air New Zealand, which is formulated as a set partitioning problem. The idea is based on the premise that every crew member should be allocated to the next task as soon as possible after completion of the previous task and the mandatory rest. By limiting the number of subsequences, the number of variables in the set partitioning problem is reduced, and the structure of the constraint matrix becomes more balanced, as explained by Ryan and Falkner [1988]. We use the idea of limited subsequences in two ways: to generate the initial set of feasible recovery duties for the branch-and-price algorithm and as an efficient column generation pricing strategy.

As shown in Section 4.1.2, the linear programming relaxation of the set partitioning problem with GUB constraints has strong integer properties. The

fractions in the optimal solution to the TDRP–LP can only occur if two or more train drivers compete for one or more train tasks in their recovery duties. This knowledge is exploited by employing a *constraint branching* strategy where at every branch some train drivers are either forced respectively forbidden to be assigned to particular train tasks. The approach to solve integer programming problems with the linear programming relaxation and column generation at every node of the branch-and-bound tree is a widely used methodology and is known as *branch-and-price* (Barnhart et al. [1998b]). The branch-and-price framework is programmed in C#.NET. The C#.NET Application Programming Interface (API) of MOSEK Optimization Software, versions 4.0 and 5.0 (www.mosek.com) is used to solve linear programming problems.

5.2 Branch-and-Price Framework

A branch-and-price algorithm is a method for solving zero–one integer programming problems using a linear programming relaxation and column generation at each node of a branch-and-bound tree. For a methodology review and applications the reader is referred to e.g. Desrosiers et al. [1995] or Barnhart et al. [1998b]. For reviews of column generation applied to integer programming problems see for example Wilhelm [2001] or Lübbecke and Desrosiers [2005]. Dynamic column generation is a useful tool for solving LP problems with a large number of columns compared to the number of rows. The idea with the column generation approach for solving LP problems is to avoid generating all variables, since only a few of them will comprise the optimal basis. The column generation method decomposes the LP problem into a *restricted master problem* and a *subproblem*. The restricted master problem (RMP) contains only a subset of the variables from the LP problem and is solved with the revised simplex method. The subproblem is the pricing problem that generates new columns to enter the basis. The dual variables of the optimal solution to the RMP are used to calculate the reduced costs of the non-basic variables. Only non-basic variables with negative reduced costs can enter the basis and improve the objective function value of a minimization linear programming problem. Therefore, for a minimization problem, the solution is optimal when no negative reduced cost columns can

be generated by the pricing problem.

If the solution to the RMP is integer, then the optimal solution to the zero–one IP problem is found. If, on the other hand, the optimal LP solution does not satisfy the integrality constraints, the integer solution is obtained by the widely employed branch-and-bound technique. A general description of the methodology for solving integer programs using branch-and-bound can be found in Wolsey [1998]. The branch-and-bound technique is an *implicit* enumeration of all feasible solutions. The feasible region of the problem is partitioned into smaller subproblems according to a certain branching strategy. Subproblems are stored in a branch-and-bound tree. In the LP-based branch-and-bound the linear programming relaxations of the subproblems are solved at each node of the tree. The algorithm terminates when all nodes in the tree are either branched or pruned. For a minimization problem, lower bounds at the nodes of the branch-and-bound tree are provided by optimal solutions to the LP-relaxations, while upper bounds are provided by feasible (integer) solutions. A node in the branch-and-bound tree is *pruned by bound* if the value of the optimal solution to the LP-relaxation at this node is larger than or equal to the global upper bound of the problem. The *global* upper bound is the best known integer solution. A node is *pruned by infeasibility* if the solution to the LP-relaxation at the node is infeasible. A node is *pruned by feasibility* if the optimal solution to the RMP of this node is integer, i.e. when the lower bound of the subproblem equals its upper bound. If the solution is integer and the objective value of the problem is equal to the *global* lower bound of the problem, the node represents the optimal solution to the IP. In all other cases the branching of nodes continues, unless an early termination of the algorithm is chosen, i.e. when the difference between the global bounds is less than a certain threshold. There is an important difference between the LP-based branch-and-bound and the branch-and-price algorithm. At any node of the branch-and-price tree, the presence of *all* non-basic variables in the restricted master problem cannot be ensured. Therefore, column generation is implemented at every node of the branch-and-price tree while respecting any restrictions imposed on the pricing problem by the branching strategy.

5.3 Linear Programming Relaxation of TDRP

The linear programming relaxation of the train driver recovery problem is formulated in (5.1)–(5.6). The problem formulation of TDRP (4.1)–(4.4) is extended with two sets of artificial variables. One artificial variable f_i is added for each train task $i \in N$ and one variable e^k for each driver $k \in K$. By adding artificial variables we ensure feasibility of TDRP–LP. A big-M cost is assigned to the artificial variables in the objective function, which ensures that $f_i, \forall i \in N$ and $e^k, \forall k \in K$ are only included in the optimal solution of the TDRP–LP if their presence is necessary for the problem feasibility. The artificial variables are used to detect which constraints would otherwise not be covered in the set partitioning problem, and hence determine the way in which the disruption neighbourhood can be expanded.

$$\text{(TDRP–LP)} \quad \text{Minimize} \sum_{k \in K} \sum_{r \in R^k} c_r^k x_r^k + \sum_{k \in K} M e^k + \sum_{i \in N} M f_i \tag{5.1}$$

$$\text{Subject to} \quad \sum_{r \in R^k} x_r^k + e^k = 1 \qquad \forall k \in K, \tag{5.2}$$

$$\sum_{k \in K} \sum_{r \in R^k} a_{ir}^k x_r^k + f_i = 1 \qquad \forall i \in N, \tag{5.3}$$

$$x_r^k \geq 0 \qquad \forall r \in R^k,\ \forall k \in K, \tag{5.4}$$

$$e^k \geq 0 \qquad \forall k \in K, \tag{5.5}$$

$$f_i \geq 0 \qquad \forall i \in N. \tag{5.6}$$

The column generation solution method for solving the TDRP–LP is illustrated in Figure 5.1. The process begins in generating an initial set of columns for a given disruption neighbourhood, as explained in Section 5.5. The restricted master problem and the subproblem are solved iteratively, using the values of dual variables corresponding to to train driver constraints (5.2) and train task constraints (5.3) to calculate the reduced costs of recovery duties. If no negative reduced cost columns can be generated by the pricing problem and no artificial variables are present in the solution to the RMP, the optimal solution to the TDRP–LP restricted by the current disruption neighbourhood has been found. If at least one artificial variable is present in the

solution, the disruption neighbourhood can be expanded by extending the recovery period for one or more train drivers or by adding other train drivers to the disruption neighbourhood. Disruption neighbourhood expansion corresponds to adding train driver and train task constraints to the problem, as described in Section 5.7.

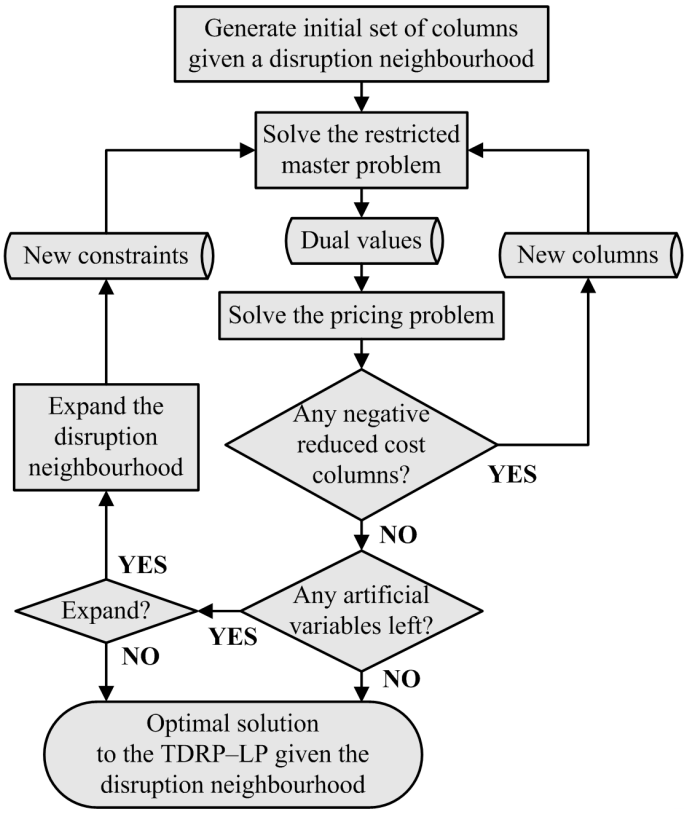


Figure 5.1: Solving the TDRP–LP to optimality with column generation.

5.3.1 Solving the Restricted Master Problem

The RMP is solved using the revised simplex method of MOSEK optimizer for linear programming problems. The simplex method was developed by George Dantzig in the 1940's and has been successfully implemented in a

large number of commercial LP solvers since then. In short, the optimal solution to a minimization linear program by the primal simplex method is found by searching through the extreme points of the feasible region of the problem, moving from one extreme point to another along the edges of the polyhedron, which defines the feasible region, in a way such that the objective function does not increase. For more information about the simplex method or the linear programming in general see e.g. Hillier and Lieberman [1995].

The performance of MOSEK optimizer is customized to this particular application by tuning available parameters. Different preprocessing steps are applied to the linear programming problem by default prior to the simplex algorithm (see MOSEK .NET API Manual, Ver.5, 2008). Preprocessing reduces the size of the LP by removing redundant constraints, eliminating free variables, removing linear dependencies. Since the problem sizes of TDRP–LP are relatively small and the optimizer is used in a "hot-start" mode during column generation, we choose to switch off the problem reduction preprocessor. Computational experiments show that by doing so a decrease of approximately 10%–12% in the running times is achieved for many problem instances. Experimenting with manual adjustments of other preprocessing parameters have not shown any computational advantage. Therefore, the default settings of the simplex optimizer are used otherwise (e.g., the default pricing strategy etc.).

5.3.2 Solving the Pricing Problem

The values of the dual variables corresponding to the train driver constraints (5.2) and the train task constraints (5.3) are used to calculate the reduced costs of recovery duties in the pricing problem of the column generation algorithm in the following way. Let λ^k be the dual variable corresponding to the k'th train driver constraint (5.2) and let π_i be the dual variable corresponding to the i'th train task constraint (5.3) in the RMP. The *reduced cost* $\bar{c}_r^k$ of the variable x_r^k is:

$$\bar{c}_r^k = c_r^k - \lambda^k - \sum_{i \in N} a_{ir}^k \pi_i, \tag{5.7}$$

where c_r^k is the cost of a feasible recovery duty $r \in R^k$ of the driver $k \in K$. The recovery duty cost c_r^k is defined in (4.6), Section 4.2.4.

The negative of value of the dual variable λ^k for $k \in K$ is set as the cost of the source vertex o^k in G^k: $c(o^k) = -\lambda^k$. Values of dual variables π_i for $i \in N^k$, with opposite signs, are set as costs on train task vertices in G^k: $c(v_i) = -\pi_i$ for each vertex $v_i \in N^k$. The cost of the sink vertex d^k remains zero: $c(d^k) = 0$. Let a resource constrained path $p = o^k \rightarrow v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_h \rightarrow d^k$ in G^k represent a feasible recovery duty $r \in R^k$ for driver $k \in K$, as described in Section 4.2.3. Let $N^p = \{v_1, v_2, ..., v_h\}$ be the set of train task vertices in p. The cost of path $c(p)$ is defined in (4.5) as a sum of vertex costs and arc costs. Using (4.5) and (4.6) we can express the reduced cost of the recovery duty r through the cost of path $c(p)$:

$$\begin{aligned} \bar{c}_r^k &= c_r^k - \lambda^k - (\sum_{i \in N^P} 1 \cdot \pi_i + \sum_{i \in N^k \setminus N^p} 0 \cdot \pi_i) \\ &= c_r^k - \lambda^k - \sum_{i \in N^p} \pi_i - 0 \\ &= c(p). \end{aligned} \tag{5.8}$$

Hence, the reduced cost of the variable representing a recovery duty for a driver k is the cost of the resource constrained path p generated on the duty graph G^k that has vertex costs equal to the negative of the relevant dual variables. The pricing problem identifies variables in the TDRP–LP with $\bar{c}_r^k < 0$ by collecting resource constrained paths in duty graphs with $c(p) < 0$.

5.4 Recovery Duty Generation Algorithm

Resource constrained paths in G^k are generated with a *duty generation algorithm* based on dynamic programming, where a label setting procedure is used. Variations of the algorithm are used for generating the initial set

of columns for the TDRP–LP and in the pricing problem of the column generation algorithm. For a detailed description of label setting algorithms the reader is referred to Ahuja et al. [1993], while efficient label setting algorithms for resource constrained shortest path problems can be found in e.g. Aneja et al. [1983], Desrochers and Soumis [1988] and Dumitrescu and Boland [2001].

5.4.1 Components of the Algorithm

Basic components of the recovery duty generation algorithm are presented in this section. Let p be a resource constrained path in the duty graph G^k, which represents a feasible recovery duty $r \in R^k$ for a driver $k \in K$. A *subpath* $p(v)$ is a part of p from the source vertex o^k to vertex $v \in \{N^k \cup d^k\}$. A subpath is feasible, if it satisfies the break feasibility condition within the time interval $[t_{\text{arr}}(o^k); t_{\text{arr}}(v)]$, where $t_{\text{arr}}(v)$ is the time of arrival of a train task v. A *label* $l(w)_i^j$ is generated for every feasible subpath $p(w), w \in \{N^k \cup d^k\}$. The subscript i of the label refers to the number of the *predecessor* label $l(v)_h^i, v \in \{o^k \cup N^k\}$. Label $l(v)_h^i$ is the predecessor label of $l(w)_i^j$ if the subpath $p(v)$ is extended along the arc (v, w): $p(w) = p(v) + (v, w)$. By storing references to predecessor labels it is possible to quickly trace any subpath $p(w)$ back to the source vertex o^k. Label $l(w)_i^j$ also stores the information about the accumulated cost of the subpath $p(w)$, the length of the break time left to be held in the recovery duty after finishing the part of the duty represented by $p(w)$, and the earliest and the latest start times of the next break. Every label $l(d^k)_i^j$ of the sink vertex d^k stores a feasible resource constrained path p from o^k to d^k, i.e. a feasible recovery duty for the driver k in the disruption neighbourhood of the problem.

The recovery duty generation label setting algorithm consists of an *initialisation step*, a *label selection step* and a *label treatment step*. During the initialisation step, an initial label $l(o^k)^0$, a list of labels L and a list of labels $L(v)$ are generated. All generated labels are stored in L, while $L(v)$ only contains labels generated for vertex v. The label selection and the label treatment steps are performed iteratively until the the algorithm terminates. The algorithm terminates either when all labels in L are treated or when a required number of paths is collected. During the selection step, a label in

L is chosen to be treated. The labels in L can be lexicographically sorted by the cost of the corresponding subpaths or other resources, e.g. the break time left to be held within the recovery duty. Then the lexicographically cheapest label is chosen for treatment. However, computational experiments with the recovery duty generation algorithm have shown that the simple first-in-first-out label selection works better for the problem instances tested in this project. During the label treatment step, a *feasibility check* and a *dominance check* are applied. The label treatment is aimed at extending the subpath of the treated label to the adjacent vertices and thereby generate new labels.

During the feasibility check of a label $l(v)_h^i$ it is determined if the subpath $p(v)$ can be extended along all outgoing arcs (v, w), i.e. if any subpath $p(w)$ is feasible with respect to the break feasibility condition. A simplified break feasibility condition check is described in Algorithm 5.1. Only if `IsFeas` = **true**, a new label $l(w)_i^j$ is generated. If `IsHeld` = **true** then the length of the break time left to be held in the recovery duty is updated by subtracting the length of the largest possible break duration on arc (v, w), and updating the earliest and the latest start times of the next break. Note that since the break resource can only be updated on break opportunity arcs, these arcs can also be called *replenishment arcs*, using the terminology of Boland et al. [2000].

Algorithm 5.1 Break feasibility condition check

`IsHeld` = **false**, `IsFeas` = **false**.
if a break is not required after $t_{\text{arr}}(v)$ **then**
  `IsFeas` = **true**.
else
  if the required break can be held on (v, w) **then**
    if it is not too late to hold the break **then**
      if it is not too early to hold the break **then**
        `IsFeas` = **true**, `IsHeld` = **true**.
      else
        if it is not too late to hold the break after $t_{\text{arr}}(w)$ **then**
          `IsFeas` = **true**.
  else
    if it is not too late to hold the break after $t_{\text{arr}}(w)$ **then**
      `IsFeas` = **true**.

A dominance check is applied to the labels belonging to the same vertex in order to reduce the number of labels to be considered in the duty generation algorithm. Label $l(v)_h^i$ *dominates* label $l(v)_b^a$ if the break time left associated with the label $l(v)_h^i$ is less than or equal to the break time left associated with the label $l(v)_b^a$ and if the cost of the subpath $p(v)$ associated with the label $l(v)_h^i$ is less than or equal to that of $l(v)_b^a$. A label is only generated and stored if no other label in $L(v)$ dominates it. Every time a new label is generated, it is also checked, if it dominates any other labels in $L(v)$. All labels dominated by the newly generated label are removed from the set.

As described in Section 5.3.2, the cost of a generated resource constrained path $c(p)$ corresponds to the reduced cost $\bar{c}_r^k$ of the corresponding variable in TDRP–LP. When the recovery duty generation algorithm is used in the pricing problem, only paths with $c(p) < 0$ are returned to the restricted master problem.

5.4.2 Enumeration of Feasible Recovery Duties

The recovery duty generation algorithm can be used for a *total enumeration* or for a *restricted enumeration* of feasible recovery duties of the train drivers in K. When duties are enumerated, the dominance check is omitted from the algorithm. The algorithm terminates when all or a number of resource constrained paths on the duty graph is collected. The total enumeration of feasible recovery duties is presented in Algorithm 5.2, while the restricted enumeration of duties by limited subsequences is described in Section 5.5.

Following the small example from previous chapters, we illustrate the total enumeration of the feasible recovery duties for the train driver assigned to Duty 2. Figure 5.2 shows all labels in the duty generation algorithm necessary to enumerate the two resource constrained paths on the duty graph G^2. According to the original schedule (see Figure 2.5, page 48) the driver assigned to Duty 2 is entitled to a 20 minutes break, which must begin no later than 17:20. The initial label $l(o^2)^0$ is therefore not extended along the arcs (o^2, v_3) and (o^2, v_4), since the break feasibility condition check returns `IsFeas` = **false**. For the same reason the label $l(v_1)_0^1$ is not extended along the arcs (v_1, v_3) and (v_1, v_4). Hence, the only feasible recovery duties for the

Algorithm 5.2 Total enumeration of feasible recovery duties on G^k

Initialisation Step:
Initialise $l(o^k)$ and L.
while all labels in L are non-treated **do**
  Label Selection Step:
  Select a non-treated label $l(v)$ from L.
  Label Treatment Step:
  for all arcs (v, w) **do**
    Perform feasibility check of $l(w)$, described in Algorithm 5.1.
    if $l(w)$ is feasible **then**
      Add $l(w)$ to L.
      if $w = d^k$ **then**
        Backtrace p from $l(d^k)$.
        Mark $l(d^k)$ as treated.
  Mark $l(v)$ as treated.

driver are represented by paths $o^2 \rightarrow v_1 \rightarrow d^2$ and $o^2 \rightarrow d^2$.

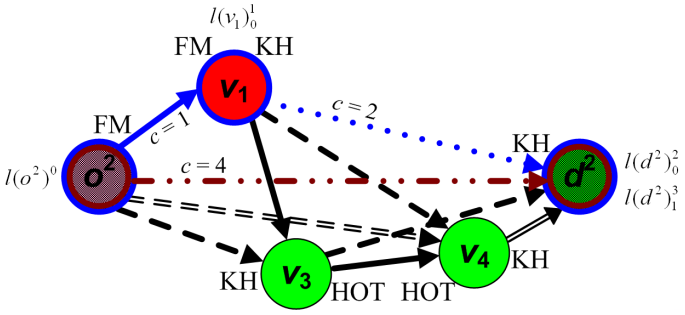


Figure 5.2: Labels in the total enumeration of feasible recovery duties on G^2.

5.4.3 Ressource Constrained Shortest Path

The resource constrained shortest path algorithm is presented in Algorithm 5.3. Due to the dominance check described in Section 5.4.1, only one label $l(d^k)$ is left in L at the termination of the algorithm. The path which is

stored in the label is the ressource constrained shortest path in G^k. In the context of the pricing problem iteration, the generated path corresponds to the non-basic variable with the most negative reduced cost among all non-basic columns in the submatrix A^k corresponding to the train driver k. When the dominance criteria is strong, i.e. if many labels can be eliminated from every $L(v)$ in the dominance check of the label setting algorithm, then the label setting algorithm is very efficient for solving the resource constrained shortest path problem.

Algorithm 5.3 Resource constrained shortest path on G^k

Initialisation Step:
Initialise $l(o^k)$, L and $L(v)$ for each $v \in \{N^k \cup d^k\}$.
while all labels in L are non-treated **do**
  Label Selection Step:
  Select a non-treated label $l(v)$ from L.
  Label Treatment Step:
  for all arcs (v, w) **do**
    Perform feasibility check of $l(w)$, described in Algorithm 5.1.
    if $l(w)$ is feasible **then**
      if $l(w)$ is not dominated by any other labels in $L(w)$ **then**
        Add $l(w)$ to L and $L(w)$.
      if $l(w)$ dominates any other labels in $L(w)$ **then**
        Remove dominated labels from $L(w)$ and L.
      if $w = d^k$ **then**
        Mark $l(d^k)$ as treated.
  Mark $l(v)$ as treated.
Backtrace p from $l(d^k)$.

5.5 Initial Set of Columns

The initial set of columns for the root node of the branch-and-bound tree could be limited to the set of artificial variables, corresponding to the unit matrix $I_{|K+N|}$. However, the information from the dual variables of the optimal solution to the RMP with such a set of columns is not very useful. In order to generate better quality dual vectors from the initial set of columns, a number of feasible recovery duties for all drivers in K is generated.

A *limited subsequences enumeration* is performed in order to generate a small number of attractive recovery duties fast for the initial set of columns. For every train task vertex $v \in V^k$ in a duty graph G^k the set of possible subsequent tasks within the disruption neighbourhood is represented by outgoing arcs of the vertex. The restricted enumeration of the feasible recovery duties in G^k by limited subsequences enumeration is therefore implemented by applying the following adjustment to the Algorithm 5.2: At every label treatment step of the algorithm a feasibility check is only applied to a *limited number* of outgoing arcs of a vertex v. The number of outgoing arcs is limited to η^k, which is a small number compared to the number of outgoing arcs from v. The number η^k is calculated individually for every duty graph G^k, since the number of outgoing arcs from vertices of different graphs differs within the same disruption neighbourhood. The outgoing arcs for a feasibility check can be chosen at random, but we prefer to force the attractive recovery duties to be included to the optimization problem at an early stage of column generation. Outgoing arcs of every vertex $v \in V^k$ are therefore sorted in an ascending order of their costs during the generation of duty graphs, described in Section 4.2.2. Hence, at most η^k cheapest outgoing arcs of every vertex are checked. The restricted enumeration with limited subsequences does, however, not ensure that *the cheapest* recovery duties are collected.

Limited subsequences enumeration with $\eta^2 = 1$ for the train driver assigned to Duty 2 is illustrated on Figure 5.3. For comparison, the total enumeration of feasible recovery duties on the same duty graph G^2 is illustrated on Figure 5.2. Compared to the total enumeration, the number of generated labels is decreased from four to three and the number of paths is decreased from two to one, the only generated path being $o^2 \rightarrow v_1 \rightarrow d^2$.

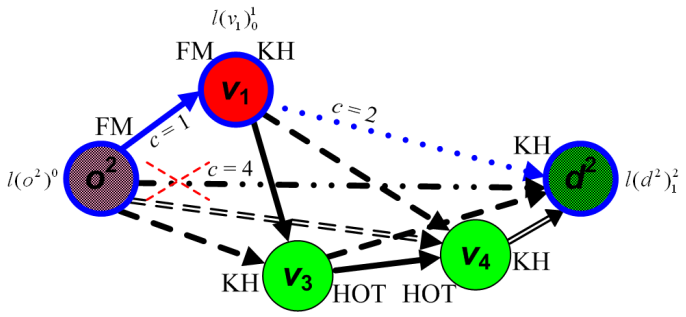


Figure 5.3: Feasible recovery duty generated with $\eta^2 = 1$.

Note that the limited subsequences enumeration with $\eta^k = 1$ generates a set of columns for every train driver $k \in K$, which composes a *totally unimodular* matrix A^k. Ryan and Falkner [1988] define a sufficient condition for the structure of a constraint matrix A, which ensures the totally unimodularity of the matrix: "if a zero–one matrix A has unique subsequence or precedence, then A is totally unimodular". A zero–one matrix A has *unique subsequence* if the rows of A can be ordered in a way that all columns with a 1 in any row i have a subsequent 1, if it exist, in a *unique* row j. For more details on unique subsequence please refer to Ryan and Foster [1981] and Ryan and Falkner [1988]. When columns are generated with $\eta^k = 1, \forall k \in K$, every row in A^k has a unique subsequence, since every vertex in the feasible duty path has a *unique* subsequent vertex, limited by the "$\eta^k = 1$"–condition in the duty enumeration algorithm. An example of a zero–one totally unimodular matrix with unique subsequences is shown on Figure 5.4. The number of the subsequent row j is shown next to every row i of the matrix. The matrix represents a constraint matrix of TDRP with the disruption neighbourhood of two train drivers (rows 1 and 2) and five train tasks (rows 3 – 7). In fact, this is a special case, where not only submatrices $A^k, k = 1, 2$ are totally unimodular, but the overall matrix A of TDRP–LP is totally unimodular.

Row i	Subseq. row j					
1	3	1	1			
2	4			1	1	1
3	5		1			
4	6			1	1	1
5	–		1			
6	7				1	1
7	–					1

Figure 5.4: Example of a totally unimodular constraint matrix of TDRP.

5.6 Pricing Strategies

The pricing strategies implemented within the column generation framework for solving the TDRP–LP are summarized in Table 5.1. All strategies employ the resource constrained shortest path algorithm, 5.3, described in Section 5.4.3.

Table 5.1: Pricing strategies.

Abbr.	Strategy	Max.Col.
FP	Full pricing of all potential columns.	1
PP	Partial pricing of one duty graph at a time.	1
MP	Multiple pricing.	$\|K\|$
SP	Limited subsequences strategy pricing.	$\|K\|$

Full pricing is the classical approach to solve the pricing problem in the context of column generation methods for solving LP problems. The full pricing strategy corresponds to the Dantzig rule (see e.g. Dantzig [1963]) of finding the entering variable in the primal simplex method, where all non-basic variables are scanned, and the one with the best reduced cost is chosen. Every iteration of the full pricing strategy **FP** returns the column with the minimum negative reduced cost, if one exists.

The full pricing strategy can be computationally expensive for the problems

with many columns. Instead, only a subset of the non-basic columns can be scanned, and the best candidate for the entering variable can be chosen. This type of the pricing strategy is called *partial pricing*. Partial pricing is static, when the same partition of columns is used, and dynamic, when the partition of columns is dynamically redefined during the solution process. We employ a static version of the partial pricing. The columns are partitioned by $k \in K$, i.e. every partition of columns contains the submatrix A^k for the train driver k. At every iteration of the partial pricing **PP** the minimum cost resource constrained path is found in one duty graph G^k, is such a path exists. If no negative reduced cost columns can be generated for a particular train driver, the next subset of columns is priced, which corresponds to recovery duties of the next train driver in the sequence.

Another alternative is *multiple pricing*. At each pricing iteration of the column generation algorithm, several candidate columns are generated. In the context of the simplex method, the pricing iteration is called a *major iteration*. When the candidate columns are added, several *minor iterations* are performed in order to optimize the current basis over the set of candidate columns. A more detailed description of different pricing strategies in the simplex algorithm can be found in e.g. Orchard-Hays [1968] and Chvátal [1983].

In the multiple pricing strategy **MP** a major iteration is performed by applying the resource constrained shortest path algorithm to all duty graphs in order to collect a set of columns with negative reduced costs, at most one column from each graph. The columns are returned to the RMP, and some minor iterations are performed by the LP-solver in order to update the basis. The limited subsequences pricing strategy **SP** is also a multiple pricing strategy, where each pricing iteration returns at most $|K|$ columns, as in **MP**. However, the duty paths are generated with a restriction: the limited subsequence filter η^k is used to reduce the number of arcs investigated at every label treatment step of Algorithm 5.3, exactly in the manner the initial set of columns is collected using a restricted enumeration algorithm, 5.2. The value of η^k is increased with a certain margin $\Delta\eta^k$ at every major iteration. Hence, every major iteration of the pricing strategy generates recovery duties with $\eta^k = \eta^k + \Delta\eta^k$, thereby allowing driver k to perform a larger number of subsequent tasks after finishing a task in the recovery duty. We call this increase of η^k the *deepening* of limited subsequences. The value of $\Delta\eta^k$ is

calculated separately for every duty graph G^k. Let $\delta^+(v)$ be the set of arcs leaving vertex $v \in V^k$ in G^k. Then $\Delta\eta^k$ is calculated as follows:

$$\Delta\eta^k = \lceil \frac{\max_{v \in V^k} |\delta^+(v)|}{\eta_{\text{part}}} \rceil, \tag{5.9}$$

where η_{part} is a predefined number to partition the maximum number of outgoing arcs. The initial set of columns is generated with $\eta^k = \eta^k_{\text{start}}$, which in the current implementation is equal to $\Delta\eta^k$. The value of η^k is increased until all outgoing arcs in the duty graph are opened for investigation, i.e. when $\eta^k = \max_{v \in V^k} |\delta^+(v)|$. When the value of η^k has reached its maximum size, the limited subsequences pricing strategy corresponds to **MP**. If the initial disruption neighbourhood is expanded by adding a new train driver k', the pricing of the duty graph $G^{k'}$ begins with $\eta^{k'}_{\text{start}}$, and then gradually increases as the pricing continues.

5.7 Implementing Expansion of Disruption Neighbourhood

Since one attempts to keep the disruption neighbourhood as small as possible, optimal solutions to some problem instances of TDRP restricted by the current disruption neighbourhood still contain non-covered train tasks or train drivers not assigned to any recovery duty. Note that when a train driver is not assigned any feasible recovery duty, it means that not even a stand-by task can be assigned to the driver in the recovery solution for the reasons described below. The presence of artificial variables e^k and f_i in TDRP–LP (see Section 5.3) makes it is easy to control the source of infeasibility of TDRP restricted by a given disruption neighbourhood.

5.7.1 Extending Recovery Period and Duty Length

If an artificial variable e^k is present in an optimal solution to TDRP–LP restricted by disruption neighbourhood, it implies that no feasible recovery

duty could be generated for the driver k within the recovery period, and the variable e^k is the only one that covers the k'th GUB row (5.2). There are two reasons why it is not possible to generate a feasible recovery duty for the driver k in the duty graph G^k. Firstly, there may be no direct path from the commencing task vertex o^k to the terminating task vertex d^k. Due to the disruption there is simply not enough time to travel from the arrival station of o^k to the departure station of d^k, not even by taxi. Secondly, a direct path exits; however, the path does not satisfy the break resource constraint. This is because either there are no arcs on the path that contain a break opportunity or the break opportunity time is too short.

If no feasible recovery duty is generated for a particular train driver k, and the terminating task vertex d^k in the driver's duty graph G^k is represented by a check-out activity, then extending the recovery period would not resolve the infeasibility. The train driver's check-out activity start time is therefore delayed with a fixed number of minutes, e.g. 10-15 minutes. The shift in the check-out activity corresponds to extending the duty length. The train driver k is then paid for the registered overtime work. If d^k is not represented by a check-out activity, the recovery period for the driver is extended with a certain time interval. This is illustrated on an example presented in Section 3.2.2. When the recovery period is extended for driver k, the train driver is assigned a new terminating task, determined by the extended end recovery time. Any train tasks in the original duty of k, which take place before the new terminating task, are added to the set of train tasks N in the disruption neighbourhood if these train tasks are not yet included in N. All duty graphs are updated in order to ensure that a vertex representing each of the newly added train tasks is included in the vertex sets V^k of $G^k, \forall k \in K$. Finally, the TDRP–LP is updated by adding train task constraints (5.3) and the corresponding artificial variables f_i' for all $i' \in N'$, where N' is the set of newly added train tasks.

5.7.2 Adding Train Drivers

If an artificial variable f_i remains present in the optimal solution to the restricted TDRP–LP, it means that no one of the drivers within the disruption neighbourhood is able to cover the train task i. In this situation there are

two possibilities. One option is to add a new train driver or a set of new train drivers to the disruption neighbourhood. Alternatively, the train task i can be left non-covered. An obvious choice is to add a train driver who is currently in reserve and who has sufficient time to cover at least one train task. Another alternative is to add a driver who is currently assigned to other train tasks and may potentially be able to cover one of the non-covered train tasks.

When the disruption neighbourhood is expanded by adding a set of train drivers K' to K, all train tasks between the commencing and the terminating tasks of every $k \in K'$ are added to the problem. Introducing a new driver k' to the disruption neighbourhood corresponds to adding an artificial variable $e^{k'}$ and a train driver constraint (5.2) for $k = k'$ to the TDRP–LP. Duty graphs are generated for all train drivers in K'. If a set of new train tasks N' is added to N in the disruption neighbourhood, all existing duty graphs are updated in order to ensure that vertices corresponding to the trains in N' are included. A set of train task constraints (5.3) and the corresponding artificial variables f_i' for all train tasks $i \in N'$ are added to TDRP–LP.

5.7.3 Implementation Details

In the current implementation of the prototype to the TDR–DSS the disruption neighbourhood is expanded only in order to restore feasibility. Following actions are allowed during disruption neighbourhood expansion if it is necessary for the solution feasibility:

- Shifting the start time of the check-out task by 15 minutes.
- Extending the recovery period of train drivers one hour at a time.
- Adding reserve train drivers.

There is a natural order in which the disruption neighbourhood is expanded, dictated by common sense and computational experiments. First, we resolve the problem infeasibility caused by not being able to generate recovery

duties. Second, the infeasibility cased by non-covered train tasks in the solution is dealt with. Computational experiments have shown that the most frequently occurring infeasibility can only be resolved by shifting the start time of the check-out task in the drivers' duties. Since no other action can be taken, it makes sense to resolve this type of infeasibility first. Next we resolve the infeasibility which requires extension of the train drivers' recovery periods. Extending recovery periods for some train drivers might in some situations help covering the non-covered train tasks, if such train tasks exist. At last, if at least one train task remains not covered in the recovery solution, reserve drivers are added to the problem. Computational experiments show that when all available reserve train drivers are added to the disruption neighbourhood in one iteration, the computational time spent on generating graph data structures for the newly added train drivers is less than the computational time required for adding one train driver per iteration. Adding all available reserve drivers does however not mean that all of them will be involved in the recovery.

The disruption neighbourhood is expanded at least until all train drivers in K have at least one feasible recovery duty each. If for some reason the disruption neighbourhood is chosen not to bo expanded further, while at least one artificial variable f_i is still present in the solution, the the optimal solution to the TDRP–LP, restricted by the chosen disruption neighbourhood, contains non-covered train tasks.

5.8 Finding Integer Solutions

As shown in Section 4.1.2, the fractions in the optimal solution to TDRP–LP can only occur across train drivers blocks of columns. Hence, if driver k and driver k' compete for the same train task i in the optimal fractional solution, it makes sense to check if the optimal integer solution could be achieved by either forcing or forbidding one of the drivers to cover i. This observation is used to choose a branching strategy for finding optimal solutions to TDRP, namely constraint branching.

Constraint branching was originally proposed by Ryan and Foster [1981]

for the set partitioning formulations of scheduling problems. Formally, let $J(s,t)$ be the set of variables in the optimal fractional solution, where each variable covers constraints s and t simultaneously. Let the *sum of fractions* $\Pi(s,t) = \sum_{j \in J(s,t)} x_j$ be the sum of solution values of the variables in the set $J(s,t)$. As shown by Ryan and Foster [1981], any optimal fractional solution contains at least one constraint pair $\{s,t\}$, for which $\Pi(s,t)$ lies strictly between zero and one (see Barnhart et al. [1998b] for a proof):

$$0 < \sum_{j \in J(s,t)} x_j < 1. \tag{5.10}$$

In constraint branching, the 1-branch forces both constraints s and t to be covered by the same variable, which is expressed through (5.11). The 0-branch is expressed through (5.12) and implies that constraint s must not be covered by the same variable as constraint t.

$$\sum_{j \in J(s,t)} x_j = 1. \tag{5.11}$$

$$\sum_{j \in J(s,t)} x_j = 0. \tag{5.12}$$

Variations of the constraint branching strategy are broadly used in the context of scheduling and routing problems. As an example, branching on *follow-ons* is used in the aircraft routing problem (see Barnhart et al. [1998a]), where on the 1-branch it is required that a sequence of flights (i,j) must be covered by the same variable, i.e. flight j must follow flight i in the optimal integer solution, while on the 0-branch flight j is not allowed to follow flight i. For other references to applications of constraint branching strategies please see e.g. Barnhart et al. [1998b].

5.8.1 Choosing Constraint Pair for Branching

In the context of TDRP, the constraint pair $\{s,t\}$ is chosen such that s is the train driver constraint (4.2) and t is a train task constraint (4.3). The

1-branch forces the train driver corresponding to s to cover the train task corresponding to t, while the 0-branch forbids the train driver corresponding to s to have the train task corresponding to t in his/her recovery duty. The next question is how to identify a {driver, train} constraint pair for branching at a particular node of the branch-and-price tree.

In the context of the airline crew pairing problem, Ryan [1992] suggests that if the constraint pair $\{s, t\}$, where s is a crew member constraint and t is a trip constraint, with the largest (i.e., closest to 1) sum of fractions $\Pi(s, t)$ is chosen, and a depth-first search of 1-branches of the branch-and-bound tree is implemented, then the 0-branches can be left unfathomed, since every 1-branch reflects the preference of crew member s for the trip t is implied by the optimal yet fractional LP solution.

The same idea is applicable to the train driver recovery problem. The larger the sum of fractions for the constraint pair $\{s, t\}$, the stronger the likelihood that the optimal solution to the RMP will assign the driver s to the train task corresponding to t. We therefore compute all sums of fractions of a fractional solution to the RMP and then choose to branch on the constraint pair with *the largest sum of fractions*: $\{s, t\} = \arg\max_{(s,t)} \Pi(s, t)$, where $\Pi(s, t) < 1$.

Using the largest sum of fractions for choosing the constraint pair for branching does, however, not make much sense from the mathematical point of view in situations where *all* sums of fractions in $\Pi(s, t)$ are equal to 0.5. In these situations the LP has no preferences towards assigning any specific train task to any particular train driver. Choosing an *arbitrary* {driver, train} constraint pair in $J(s, t)$ for branching and continue with the depth-first search is one alternative. Another alternative is to switch to the *best-first search* of the tree as soon as the first integer solution is found by the depth-first search. This might be helpful in situations where the optimal integer solution is the part of the right side (the 0-branches) of the tree, so choosing the branch with best lower bound can lead to the optimal integer solution quicker than the depth-first search.

Unfortunately, none of the alternative ways to choose a constraint pair for branching has shown any definite positive trends, performing randomly when applied to different test instances. Computational experiments show that only a very few test instances produce fractional solutions in the root node

of the branch-and-price tree, and optimal integer solutions are often achieved within a very few iterations of the branch-and-price algorithm. Implementation of more sophisticated branching rules has therefore shown to be superfluous.

5.8.2 Implementing Constraint Branching

In the Restricted Master Problem

Branching constraints (5.11) and (5.12) are not explicitly added to the RMP. Instead, in any 1-branch node, the variables in the RMP which cover either the driver constraint s or the train constraint t, but not both, are removed from the minimization LP problem by setting the upper bounds of the variables to zero. In any 0-branch node, the upper bounds of all variables which cover both constraints s and t simultaneously are set to zero. When no extra constraints are added to the RMP, the number of dual variables remains the same, and the pricing problem is not destroyed in any node of the branch-and-price tree.

In the Subproblem

The branching restrictions must also be applied to the column generator of the pricing problem. The implemented branching strategy involves forcing and forbidding some train tasks to be present in the recovery duties of some train drivers. We therefore generate a set N^k_{force} of *forced* train tasks and a set N^k_{forbid} of *forbidden* train tasks for every driver $k \in K$. Both sets are updated at each node of the branch-and-price tree, and the sets are inherited from parent nodes to child nodes. On any 1-branch, a train task i is added to the set N^k_{force} for a driver k, if i and k correspond to the constraint pair $\{s, t\}$ chosen for branching. At the same time, the train task i is added to the set $N^{k'}_{\text{forbid}}$ for all other drivers $k' \in K$. On any 0-branch node, a train task i is added to the set N^k_{forbid} for a driver k, if i and k correspond to the constraint pair $\{s, t\}$. Note that as the branching continues, the same train

driver k might have more than one train task in both sets, but not the same task in both sets.

In order to ensure that the subproblem only generates columns, which satisfy branching constraints, a few adjustments are made to Algorithms 5.2 and 5.3, described in Section 5.4. The adjustments are only applied in the pricing problems of the child nodes in the branch-and-price tree, since $N^k_{\text{forbid}} = \varnothing$ and $N^k_{\text{force}} = \varnothing$ for all $k \in K$ in the root node of the tree.

It is a straight forward task to ensure that any train task $i \in N^k_{\text{forbid}}$ must not appear in any feasible duty of the train driver k. The *forbidding check* ensures that during the label treatment step of any label $l(v)$ in the duty generation algorithm applied to the duty graph G^k, a outgoing arc (v, w) of the vertex v is only examined if the train task represented by the vertex w does not belong to the set of forbidden train tasks N^k_{forbid} for the driver k. This requirement ensures that all ingoing and outgoing arcs of vertices represented by train tasks in N^k_{forbid} are not considered in the duty generation algorithm.

The other branching restriction, which requires that all train tasks in N^k_{force} must appear in every feasible recovery duty of driver k, is computationally more expensive. We express this restriction by adding an extra requirement to the resource constrained paths, which represent feasible recovery duties of the driver k. Train tasks in N^k_{force} are sorted by ascending departure times and added to every label in the recovery duty generation algorithm on G^k. Let $i, i', i'', ..., i^h$ be the train tasks in N^k_{force}, sorted in an ascending order of departure times. Let $\texttt{IsCover}(i)^a$=**true** if the subpath $p(v)$ corresponding to the label $l(v)^a$ contains vertex i, and $\texttt{IsCover}(i)^a$=**false** otherwise. The following dominance criteria can ensure that only feasible recovery duties containing all train tasks in N^k_{force} are returned to the RMP after pricing graph G^k: Label $l(v)^a$ *dominates by branching* label $l(v)^b$, if and only if the sequence $\{\texttt{IsCover}(i)^a, \texttt{IsCover}(i')^a, ..., \texttt{IsCover}(i^h)^a\}$ is lexicographically larger than the sequence $\{\texttt{IsCover}(i)^b, \texttt{IsCover}(i')^b, ..., \texttt{IsCover}(i^h)^b\}$, i.e., $\{\texttt{IsCover}(i)^a > \texttt{IsCover}(i)^b\}$ OR $\{\texttt{IsCover}(i)^a = \texttt{IsCover}(i)^b$ AND $\texttt{IsCover}(i')^a > \texttt{IsCover}(i')^b\}$ OR ... OR $\{\texttt{IsCover}(i^{h-1})^a = \texttt{IsCover}(i^{h-1})^b$ AND $\texttt{IsCover}(i^h)^a > \texttt{IsCover}(i^h)^b\}$. At least one feasible recovery duty r^k which contains *all* train tasks in N^k_{force} is already present in the basis of the RMP, hence there exists at least one feasible duty path in G^k, which

contains vertices corresponding to all train tasks in N^k_{force}, i.e. for which `IsCover`(i)=**true** for all $i \in N^k_{\text{force}}$. The total enumeration and the shortest path algorithms adjusted to be used in the subproblem of other nodes in the branch-and-pricing tree than the root node are presented in Algorithms 5.4 and 5.5.

Algorithm 5.4 Enumeration of feasible recovery duties with $\bar{c}^k_r$ on G^k.

Initialization Step:
Initialize $l(o^k)$ and L.
while all labels in L are non-treated **do**
  Label Selection Step:
  Select a non-treated label $l(v)$ from L.
  Label Treatment Step:
  for all arcs (v, w) **do**
    if $w \notin N^k_{\text{forbid}}$ **then**
      Perform feasibility check of $l(w)$, described in Algorithm 5.1.
      if $l(w)$ is feasible **then**
        if $l(w)$ is not *dominated by branching* by any other labels in $L(w)$ **then**
          Add $l(w)$ to L.
          if $l(w)$ *dominates by branching* any other labels in $L(w)$ **then**
            Remove *dominated by branching* labels from $L(w)$ and L.
          if $w = d^k$ **then**
            Backtrace p from $l(d^k)$ if $c(p) < 0$.
            Mark $l(d^k)$ as treated.
  Mark $l(v)$ as treated.

Algorithm 5.5 Resource constrained shortest path on G^k used in the pricing problem.

Initialization Step:
Initialize $l(o^k)$, L and $L(v)$ for each $v \in \{N^k \cup d^k\}$.
while all labels in L are non-treated **do**
  Label Selection Step:
  Select a non-treated label $l(v)$ from L.
  Label Treatment Step:
  for all arcs (v, w) **do**
    if $w \notin N^k_{\text{forbid}}$ **then**
      Perform feasibility check of $l(w)$, described in Algorithm 5.1.
      if $l(w)$ is feasible **then**
        if $l(w)$ is not *dominated by branching* by any other labels in $L(w)$ **then**
          if $l(w)$ is not dominated by any other labels in $L(w)$ **then**
            Add $l(w)$ to L and $L(w)$.
            if $l(w)$ *dominates by branching* any other labels in $L(w)$ **then**
              Remove *dominated by branching* labels from $L(w)$ and L.
            if $l(w)$ dominates any other labels in $L(w)$ **then**
              Remove dominated labels from $L(w)$ and L.
            if $w = d^k$ **then**
              Mark $l(d^k)$ as treated.
  Mark $l(v)$ as treated.
Backtrace p from $l(d^k)$ if $c(p) < 0$.

Chapter 6

Computational Experiments

This chapter presents computational results of the recent test experiments with the TDR–DSS prototype. Test data stems from the real-life operations of S-tog. Preliminary test results can be found in Rezanova and Ryan [2006] (schedule from year 2005 and simulated train line cancellations) and Rezanova and Ryan [2009] (schedule from year 2007 and real-life timetable disruptions) and are therefore not described in this chapter.

We present disruptions which occurred during one day of S-tog's operations as a consequence of a failure in a railroad switch on the S-train network. Using the real-life data, we generate 42 test instances with different sizes of disruption neighbourhoods. These instances are used to test the efficiency of the solution method with respect to the computational times. The most efficient solution strategy is used to solve the rolling time horizon test scenarios, which show how S-tog's train driver schedule disturbed by the particular disrupted situation can be recovered.

6.1 Test Data

Like in the majority of scientific projects engaged with developing prototypes for real-life applications, a significant amount of time during this thesis was spent on data processing and bringing the system as close as possible to S-tog's operations. As described in Section 3.3, the data necessary for solving TDRP consists of three parts: the train driver schedule, the timetable and the disruption records. Historical data from S-tog's operations from 23 January 2007 is used to generate all test scenarios presented in this chapter. During that particular day the operations on the S-train network were severely disrupted as a consequence of a dysfunction of a railroad switch at København H junction. According to S-tog's records, the defect in the switch occurred around noon, and caused many train delays. Even though the switch was repaired within the following two hours, the train delays cased by the dysfunction propagated throughout the network, and the operations were more or less disrupted until the rest of the day.

The available disruption records contain information about the changes in the timetable caused by the disruption. Data contains the scheduled, the actual and the cancelled departures and arrivals of all trains at all stations of the S-train network during the day. Table 6.1 shows the disruption records for the train number 20149 of line **B** from Høje Tåstrup station (HTÅ) to Holte station (HOT). Column "Arr/Dep" specifies if the times in the third and the forth columns are given for the arrival ("Ankomst") or the departure of the train ("Afgang"). The third and the forth columns of the table present the *scheduled* and the *actual* departure/arrival times, respectively. According to these records, the train departed with a 5 minutes delay from HTÅ. The departure delay from the terminal station resulted in a 17 minutes arrival delay at the central station København H (KH). The last column of the table shows that all departures between Lyngby station (LY) and Holte (HOT) were cancelled. It means that the train was re-routed by turning back at LY before it reached the terminal station HOT on the northern part of the line. Premature turning is one of a commonly used timetable recovery strategies at the S-train network described in Section 2.2.2. A train can be turned at a station, where there is a possibility for an unscheduled train to occupy a supplementary platform without disturbing the incoming and outgoing traffic. In the recovery schedule of the TDR–DSS prototype the train nr.

20149 is represented by two train tasks: a train task from HTÅ to KH, and a train task from KH to LY. The two train tasks are registered as being delayed and re-routed, respectively.

Table 6.1: Disruption data representation.

TrainNr	Station	SchedTime	ActualTime	Arr/Dep	CancDep
20149	HTÅ	16:00:00	16:05:13	Afgang	
20149	TÅ	16:02:10	16:07:23	Ankomst	
20149	ALB	16:05:10	16:10:08	Ankomst	
20149	GL	16:08:10	16:13:01	Ankomst	
20149	BØT	16:10:40	16:15:57	Ankomst	
20149	RDO	16:12:40	16:17:37	Ankomst	
20149	HIT	16:14:40	16:19:11	Ankomst	
20149	DAH	16:16:10	16:21:08	Ankomst	
20149	VAL	16:18:40	16:23:52	Ankomst	
20149	AV	16:20:50	16:26:19	Ankomst	
20149	DBT	16:23:20	16:30:45	Ankomst	
20149	KH	16:26:00	16:32:32	Ankomst	
20149	VPT	16:28:10	16:36:00	Ankomst	
20149	KN	16:30:00	16:37:43	Ankomst	
20149	KK	16:33:00	16:49:44	Ankomst	
20149	NHT	16:35:20	16:51:53	Ankomst	
20149	SAM	16:37:10	16:53:48	Ankomst	
20149	VAL	16:40:00	16:56:01	Ankomst	
20149	BFT	16:42:20	16:59:07	Ankomst	
20149	GJ	16:44:20	17:00:33	Ankomst	
20149	JÆT	16:46:10	17:02:50	Ankomst	
20149	LY	16:48:10	16:48:10	Ankomst	J
20149	SFT	16:50:50	16:50:50	Ankomst	J
20149	VIR	16:52:50	16:52:50	Ankomst	J
20149	HOT	16:55:00	16:55:00	Ankomst	J

Unfortunately, we did not manage to obtain the actual train driver schedule from the particular day of S-tog's operations. Only the planned train driver schedule was available for testing, without the registered changes applied to the train driver duties as a result of recovery from disruptions. It means that we cannot compare the recovery solutions employed by the train driver dispatchers during that day of operations to the solutions which can be achieved

by the implemented prototype to TDR–DSS. Note that even if the actual train driver schedule was available, it would not have been possible to restore the *exact* sequence of events for the purpose of comparison. In order to see how the train driver duties were changing as a consequence of recovery decisions made by the train driver dispatcher, it is also necessary to know exactly how much information about disrupted trains was available to the dispatcher at any given time during the day. This information also includes train driver absences and delays for reasons other than train arrival delays, if there were any. As described in Section 3.3.1, we have implemented driver delays and absences as a part of the prototype for TDR–DSS. These disruptions to the train driver schedule were only tested on simulated disruptions during preliminary tests, before the real-life disruption data was obtained.

Table 6.2: Arc costs for test purposes.

Arc type	Cost	Arc type	Cost
Original	0.00	Taxi	300.00
InDuty	1.00	Pas	80.00
InDutyPas	50.00	TrainTight	20.00
Immediate	5.00	TrainChange	20.00
Break	10.00	TrainIdle	20.00
BreakPas	80.00	Relief	15.00
Reserve	10.00	ReliefIdle	15.00
TrainKH	100.00		

As described in Section 4.2.4, the cost of each recovery duty is the sum of arc costs of the corresponding resource constrained path in the duty graph G^k, and the recovery objectives can therefore be expressed through the costs of arcs. Table 6.2 shows an overview of arc costs used for computational experiments. The description of arc types is presented in Table 4.4 in Section 4.2.4.

6.2 Testing the Solution Method

6.2.1 Generation of Test Instances

The set of instances for testing the prototype is generated in a following way: We collect a set of trains N^D, which run during the time interval of 5 minutes from the recovery start time. We assume the train driver dispatcher has only limited information available. It is assumed that the actual delay information is only available for the train tasks in N^D. This assumption is restricted compared to the real-life operations, since the dispatchers are able to predict delay propagations in the network to some extent. Duties of train drivers assigned to the disrupted (i.e. delayed, re-routed or cancelled) trains in N^D are included in the initial set of duties K of the disruption neighbourhood. It is furthermore assumed that the information about re-routings and cancellations of train tasks is available in TDR–DSS for the rest of the chosen recovery period, if and only if at least one of the trains in N^D is re-routed or cancelled. In this case the duties of drivers assigned to re-routed and/or cancelled train tasks are also included in the initial disruption neighbourhood, even if the commencing train tasks of these drivers are not disrupted. This assumption is close to the real-life information availability, where decisions for re-routings and cancellations are made by the network traffic controllers for some time in the future. Train tasks assigned to the duties in K within the recovery period represent the set of train tasks N of the initial disruption neighbourhood. No other train drivers or train tasks are included in the initial disruption neighbourhood.

Test scenarios shown in Table 6.3 are generated for six recovery period durations: 1, 1.5, 2, 2.5, 3 and 3.5 hours. The earliest recovery start time is chosen to be 12:00, which is approximately the time when the disruption occurred. Since the longest train task in the S-tog timetable lasts approximately 50 minutes, recovery periods of less than 1 hour are not appropriate. On the other hand, recovery periods of more than 3 hours do not make sense in the proposed framework, since the train driver recovery problem must be solved with a rolling time horizon. Every test instance is, however, not generated with a rolling horizon, but is *independent* of the previous recovery solutions. Hence it is assumed that nothing in the train driver schedule was changed

prior to every problem instance generation. Such independent scenarios are more difficult to recover when the disruption propagates. We choose these instances in order to test the prototype on scenarios of different severity. In the real-life such situations are unlikely to occur, since the train driver dispatcher starts to recover the schedule as soon as the information about the disrupted duties becomes available. Following notation is used to provide details about the test instances in Table 6.3:

- **RecovPer** is the recovery period of the test instance,
- **Dur** is the duration of the recovery period,
- **#Del** is the number of delayed train tasks in the initial disruption neighbourhood,
- **#Turn** is the number of re-routed train tasks in the initial disruption neighbourhood,
- **#Can** is the number of cancelled train tasks in the initial disruption neighbourhood,
- **Init**$|K|$ is the number of train drivers in the initial disruption neighbourhood,
- **Init**$|N|$ is the number of train tasks in the initial disruption neighbourhood,
- **Time** is the computational time in seconds for generating the test instance, including generation of duty graphs and other data representations. Computational times indicate amounts of time the associated function in the C# program spends utilizing the CPU of a Pentium 4 PC, 3.40 GHz with 1 GB RAM.

Table 6.3: Test instances.

ID	RecovPer	Dur	#Del	#Turn	#Can	Init$\|K\|$	Init$\|N\|$	Time
S11	12:00–13:00	01:00	12	0	0	12	8	0.11
S12	12:00–13:30	01:30	12	0	0	12	12	0.11
S13	12:00–14:00	02:00	12	0	0	12	20	0.17
S14	12:00–14:30	02:30	12	0	0	12	28	0.25
S15	12:00–15:00	03:00	12	0	0	12	35	0.33
S16	12:00–15:30	03:30	12	0	0	12	41	0.41
S21	12:15–13:15	01:00	17	0	0	17	13	0.17
S22	12:15–13:45	01:30	17	0	0	17	21	0.25
S23	12:15–14:15	02:00	17	0	0	17	29	0.34
S24	12:15–14:45	02:30	17	0	0	17	38	0.45
S25	12:15–15:15	03:00	17	0	0	17	40	0.50
S26	12:15–15:45	03:30	17	0	0	17	46	0.59
S31	12:30–13:30	01:00	22	2	3	22	11	0.17
S32	12:30–14:00	01:30	23	9	5	26	30	0.44
S33	12:30–14:30	02:00	23	13	18	35	54	1.13
S34	12:30–15:00	02:30	23	19	39	45	81	2.36
S35	12:30–15:30	03:00	23	27	56	55	120	5.11
S36	12:30–16:00	03:30	23	29	67	60	151	8.13
S41	12:45–13:45	01:00	28	6	4	30	17	0.30
S42	12:45–14:15	01:30	30	12	10	34	36	0.67
S43	12:45–14:45	02:00	30	15	31	44	58	1.45
S44	12:45–15:15	02:30	30	24	51	57	94	3.52
S45	12:45–15:45	03:00	30	29	63	64	131	6.67
S46	12:45–16:15	03:30	30	32	74	69	169	10.94
S51	13:00–14:00	01:00	24	8	5	30	14	0.25
S52	13:00–14:30	01:30	26	13	18	38	31	0.59
S53	13:00–15:00	02:00	26	19	39	47	56	1.45
S54	13:00–15:30	02:30	26	27	56	56	88	3.16
S55	13:00–16:00	03:00	26	29	67	61	118	5.41
S56	13:00–16:30	03:30	26	33	82	68	160	9.86
S61	13:15–14:15	01:00	37	8	10	40	16	0.36
S62	13:15–14:45	01:30	40	14	31	49	42	1.20
S63	13:15–15:15	02:00	40	23	51	59	70	2.41
S64	13:15–15:45	02:30	40	28	63	66	101	4.44
S65	13:15–16:15	03:00	40	31	74	71	133	7.31
S66	13:15–16:45	03:30	40	32	91	75	165	11.11
S71	13:30–14:30	01:00	38	9	18	44	26	0.53
S72	13:30–15:00	01:30	38	15	39	52	46	1.23
S73	13:30–15:30	02:00	38	23	56	60	75	2.63
S74	13:30–16:00	02:30	38	25	67	63	101	4.20
S75	13:30–16:30	03:00	38	29	82	68	132	6.91
S76	13:30–17:00	03:30	38	32	104	72	166	10.81

6.2.2 Generation of Initial Set of Columns

The initial set of columns is generated with a limited subsequences strategy, as described in Section 5.5. The value of η^k_{start} determines how many arcs are considered during the restricted enumeration of feasible recovery duties. This number is defined individually for each duty graph G^k. We set $\eta^k_{\text{start}} = \Delta\eta^k$, which is calculated from equation (5.9) on page 114. Since the value of $\Delta\eta^k$ depends on the predetermined value of parameter η_{part}, we test the branch-and-price algorithm with different values of η_{part}. Tests are run using all generated test instances with the multiple pricing strategy **MP**, described in Section 5.6. We choose to test five values of η_{part}: 1, 5, 10, 15 and 20. The value of $\eta_{\text{part}} = 1$ corresponds to an a priori generation of feasible recovery duties within the initial disruption neighbourhood, while the value of $\eta_{\text{part}} = 20$ makes the limited subsequences strategy very strict, initially generating very few columns. Table 6.4 shows all test results, using following notation:

- **N** is the number of nodes in the branch-and-price tree,
- **V** is the number of columns in the initial set generated with the tested value of η_{part},
- **T** is the computational time in seconds for solving the problem instance to optimality with the tested value of η_{part}.

The computational times for running test instances with $\eta_{\text{part}} = 1, 5, 10, 15$ and 20 are compared and the best value of the parameter is chosen for further tests. We compare the number of times a test series with a particular value of η_{part} outperforms every other of the five test series with respect to the computational times, as well as in how many test instances the tested series outperforms all remaining ones. Comparisons are presented in in Appendix E. According to results, the differences in the computational times are insignificant for all smaller instances, i.e. for the test instances S11–S35, and Si1–Si4 for $i = 4, 5, 6$. The a priori generation of the initial set of columns performs worst, especially for the large instances. As the size of the initial disruption neighbourhood grows, the number of feasible recovery duties becomes larger. As an example, the computational time for running the test instance S66 with $\eta_{\text{part}} = 1$ is 32.42 sec, compared to 11.63 sec with $\eta_{\text{part}} = 15$. We have also conducted a few sample experiments of the a priori

generation of columns. They show that that when reserve train drivers are included in the initial disruption neighbourhood, the number of a priori generated columns with $\eta_{\text{part}} = 1$ for exceeds 1.5–1.7 million some of the larger test instances, which results in running times of several minutes. The a priori generation of columns for TDRP is therefore not appropriate to use in the real-time recovery applications.

Comparing results, we conclude that the value of η_{part} must not be either too large or too small, i.e. the number of initially generated columns must be of a moderate size. Test series with $\eta_{\text{part}} = 5$ and $\eta_{\text{part}} = 20$ perform worse than that of $\eta_{\text{part}} = 10$ and $\eta_{\text{part}} = 15$. We choose the value of $\eta_{\text{part}} = 15$ for further testing. With this value of η_{part} the number of initially generated columns leads to a good initial basis for TDRP–LP, and the branch-and-price algorithm converges faster.

Table 6.4: Testing the value of η_{part}.

	$\eta_{\text{part}} = 1$			$\eta_{\text{part}} = 5$			$\eta_{\text{part}} = 10$			$\eta_{\text{part}} = 15$			$\eta_{\text{part}} = 20$		
ID	N	V	T	N	V	T	N	V	T	N	V	T	N	V	T
S11	1	16	0.38	1	11	0.38	1	11	0.36	1	11	0.33	1	11	0.34
S12	1	25	0.39	1	14	0.42	1	11	0.39	1	11	0.41	1	11	0.41
S13	1	69	0.41	1	22	0.39	1	12	0.41	1	11	0.41	1	11	0.44
S14	1	163	0.48	1	62	0.47	1	28	0.61	1	21	0.56	1	14	0.50
S15	1	402	0.58	1	114	0.59	1	51	0.75	1	37	0.70	1	26	0.61
S16	1	1105	0.78	1	222	0.78	1	91	0.97	1	53	0.94	1	41	0.97
S21	1	29	0.55	1	15	0.45	1	15	0.42	1	15	0.44	1	15	0.42
S22	1	65	0.59	1	27	0.52	1	20	0.48	1	15	0.50	1	15	0.50
S23	1	120	0.47	1	46	0.48	1	25	0.42	1	15	0.53	1	15	0.55
S24	1	364	0.58	1	139	0.58	1	66	0.69	1	40	0.67	1	25	0.78
S25	1	394	0.58	1	140	0.67	1	69	0.75	1	45	0.67	1	26	0.70
S26	1	882	0.78	1	275	0.89	1	125	1.11	1	72	0.91	1	50	1.14
S31	1	30	0.81	1	15	0.77	1	15	0.72	1	15	0.70	1	15	0.72
S32	1	150	0.81	1	58	0.83	1	37	0.75	1	25	0.80	1	25	0.83
S33	1	1185	1.27	1	266	1.23	1	122	1.28	1	89	1.33	1	63	1.34
S34	1	3551	1.39	1	830	1.34	1	409	1.38	1	256	1.42	1	196	1.53
S35	1	22765	4.91	1	3564	4.30	1	1539	4.11	1	916	3.75	1	528	3.72
S36	1	118872	22.53	7	12257	20.36	3	4359	12.72	13	2388	29.22	3	1562	29.22
S41	1	58	1.15	1	25	1.17	1	21	1.11	1	19	1.11	1	19	1.11
S42	1	365	1.20	1	106	1.31	1	57	1.34	1	44	1.50	1	33	1.50
S43	1	1211	1.70	1	281	1.95	1	144	1.89	1	100	1.86	1	77	1.86
S44	1	5009	4.33	1	1207	4.31	1	533	4.31	1	335	4.13	1	229	4.13
S45	3	46189	14.38	1	5996	5.50	1	2334	5.77	1	1186	5.38	3	772	5.38
S46	1	196324	29.73	1	17614	14.47	1	6353	14.11	1	3339	15.31	1	2010	15.31
S51	1	46	0.86	1	22	0.86	1	18	0.86	1	18	0.86	1	18	0.86
S52	1	306	1.16	1	88	1.25	1	49	1.11	1	42	1.16	1	34	1.16
S53	1	1222	1.97	1	338	2.19	3	160	2.53	3	103	2.44	1	84	2.44
S54	1	4201	2.62	1	1137	2.11	1	509	2.11	1	284	2.11	1	220	2.11
S55	1	42769	6.55	1	5190	4.23	1	1963	4.44	1	967	4.15	1	624	4.45
S56	1	212278	35.80	1	18874	15.27	1	6285	11.69	1	3138	12.44	1	1966	13.72
S61	1	42	1.34	1	26	1.30	1	23	1.17	1	22	1.16	1	22	1.17
S62	1	424	1.72	1	139	1.75	1	81	1.83	1	54	1.73	1	46	1.75
S63	1	1692	3.23	1	536	3.31	1	226	3.44	1	132	3.20	1	106	3.33
S64	1	8140	2.75	1	1828	3.17	1	709	2.59	1	388	2.75	1	272	2.75
S65	1	35320	6.94	1	6070	5.80	1	2221	6.16	1	1040	5.16	1	698	5.91
S66	1	254413	32.42	1	20452	12.70	1	6482	12.17	1	3278	11.63	1	1901	12.98
S71	1	85	1.66	1	36	1.72	1	29	1.75	1	27	1.75	1	27	1.73
S72	1	432	2.31	1	136	2.20	1	72	2.20	1	52	2.28	1	50	2.33
S73	1	1925	3.34	1	638	3.66	1	294	3.52	1	181	3.27	1	129	3.59
S74	1	8101	3.93	1	1786	3.23	1	781	2.94	1	461	2.92	1	333	2.98
S75	1	30828	6.64	1	5911	6.27	1	2078	5.63	1	1014	6.27	1	663	5.67
S76	3	271099	77.28	1	26385	16.94	1	9186	15.42	1	4690	12.72	1	2562	13.38

6.2.3 Effectiveness of Pricing Strategies

We compare the four pricing strategies implemented in the branch-and-price algorithm. For all tests the initial sets of columns are generating using the limited subsequences strategy with $\eta_{\text{part}} = 15$, which is found best suitable, as described in Section 6.2.2. The performance of the pricing strategies is tested by comparing the computational times required to achieve optimal solutions and the number of pricing iterations. We compare the pricing strategies similarly to the test comparisons of η_{part} in Section 6.2.2, and comparison results are presented in Appendix F. Table 6.5 shows computational experiments with different pricing strategies. The following notation is used:

- **N** is the number of nodes in the branch-and-price tree,
- **It** is the number of pricing iterations,
- **V** is the number of variables generated in the branch-and-price algorithm,
- **T** is the computational time in seconds for solving the problem instance to optimality with the tested pricing strategy,
- **MP** is the multiple pricing strategy,
- **SP** is the limited subsequences pricing strategy,
- **PP** is the partial pricing strategy,
- **FP** is the full pricing strategy.

The full pricing strategy **FP** performs worst, particularly on the larger test instances. The partial pricing **PP** is only slightly better than **FP**. The multiple pricing **MP** and the limited subsequences pricing **SP** perform well, the multiple pricing being the evident winner among the four pricing strategies. In **FP** and **PP** only one column is added to the RMP after each pricing iteration. Therefore, a larger number of iterations is required to finish column generation. Every pricing iteration is computationally more expensive than several iterations of the revised simplex algorithm employed in MOSEK LP-solver. Hence, when several good columns are returned to the restricted master problem, the algorithm converges faster. We therefore choose the multiple pricing strategy **MP** for further computational experiments.

Table 6.5: Testing pricing strategies.

	MP			**SP**			**PP**			**FP**		
ID	N	It	T	N	It	T	N	It	T	N	It	T
S11	1	10	0.33	1	10	0.33	1	28	0.56	1	24	0.52
S12	1	12	0.41	1	14	0.42	1	40	0.70	1	39	0.75
S13	1	14	0.41	1	15	0.44	1	56	1.20	1	55	1.20
S14	1	16	0.56	1	15	0.47	1	87	1.25	1	78	1.75
S15	1	20	0.70	1	24	0.75	1	114	2.42	1	114	2.61
S16	1	33	0.94	1	22	0.80	1	129	2.91	1	179	4.36
S21	1	10	0.44	1	11	0.44	1	36	0.92	1	41	1.31
S22	1	13	0.50	1	17	0.56	1	59	1.33	1	54	1.28
S23	1	17	0.53	1	18	0.55	1	78	1.14	1	81	1.20
S24	1	21	0.67	1	21	0.73	1	112	1.56	1	134	2.20
S25	1	16	0.67	1	22	0.77	1	98	1.67	1	111	1.98
S26	1	22	0.91	1	29	1.13	1	181	2.53	1	216	5.38
S31	1	18	0.70	1	19	0.73	1	72	1.67	1	73	1.73
S32	1	18	0.80	1	26	0.95	1	87	2.11	1	93	2.27
S33	1	26	1.33	1	27	1.34	1	149	2.66	1	188	4.52
S34	1	21	1.42	1	32	1.78	1	244	4.55	1	352	13.14
S35	1	28	3.75	1	29	3.20	1	435	12.00	1	700	43.55
S36	13	99	29.22	3	51	11.91	1	555	39.93	3	1138	133.47
S41	1	24	1.11	1	26	1.20	3	122	3.20	1	102	2.67
S42	1	27	1.50	1	30	1.52	1	112	3.14	1	98	2.27
S43	1	27	1.86	1	35	1.91	1	178	3.66	1	215	5.88
S44	1	31	4.13	1	41	4.27	1	333	8.73	1	475	18.17
S45	1	33	5.38	3	50	7.63	1	490	16.16	1	857	61.67
S46	1	48	15.31	1	40	11.72	1	636	46.42	1	1186	175.30
S51	1	18	0.86	1	22	0.92	1	84	1.70	1	79	2.14
S52	1	23	1.16	1	32	1.34	1	106	2.48	1	124	2.36
S53	3	38	2.44	1	39	2.23	1	223	4.44	1	257	7.11
S54	1	21	2.11	1	29	2.27	1	249	7.95	1	373	11.50
S55	1	24	4.15	1	35	4.22	1	408	12.77	1	548	36.66
S56	1	37	12.44	1	39	11.56	1	625	53.67	1	1017	164.69
S61	1	18	1.16	1	23	1.30	1	101	2.77	1	82	2.53
S62	1	23	1.73	3	42	2.13	1	173	3.50	1	151	3.55
S63	1	26	3.20	1	41	3.73	1	250	6.19	1	327	9.61
S64	1	24	2.75	1	38	3.61	1	332	10.91	1	390	14.56
S65	1	28	5.16	1	41	5.61	1	436	15.27	1	690	42.78
S66	1	34	11.63	1	47	12.45	1	646	36.59	1	998	139.59
S71	1	29	1.75	1	31	1.89	1	121	3.70	1	117	3.75
S72	1	33	2.28	1	37	2.27	1	197	4.14	1	265	5.69
S73	1	26	3.27	1	39	3.80	1	235	6.31	1	286	10.50
S74	1	24	2.92	1	39	3.48	1	285	11.17	1	393	18.81
S75	1	33	6.27	1	37	5.31	1	495	23.16	1	694	43.48
S76	1	34	12.72	1	46	14.13	1	663	44.81	1	1243	156.58

6.2.4 Early Termination of Branch-and-Price

Among all presented series of tests there are only a very few problem instances that require branching. All other instances are solved to optimality in the root node of the branch-and-price tree. In fact, among the 336 runs presented so far, the optimal solution is found in the root node of the branch-and-price tree in 322 runs, i.e. in ca. 96% of al test runs.

As argued previously, the feasibility of the recovery solution is more important than optimality. The employed constraint branching strategy forces the "most suitable" train driver to cover a train task, which is covered by more than one driver in the optimal fractional solution to the TDRP–LP. Combined with a depth-first search of the branch-and-price tree, the employed branch-and-price algorithm searches for a good feasible solution by looking through the left size (the 1-branches) of the tree first. We ague that the termination of the branch-and-price algorithm as soon as the first integer solution is found is sufficient for finding a solution which is close to optimal. To support this, we implement the early termination of the branch-and-price algorithm and compare results to the optimal solutions. These test cases are presented in Table 6.6, using following notation:

- **PrSt** is the pricing strategy used in the algorithm,
- $\eta_{\mathbf{part}}$ is the value of η_{part} used to generate initial set of columns,
- **App** is the solution approach for finding integer solutions. **Opt**: the test instance is solved to optimality with the depth-first search of the branch-and-price tree. **Feas**: the algorithm terminates as soon as the first feasible integer solution is found by the dept-first search,
- **#N** is the number of solved nodes in the branch-and-price tree, including the root node,
- **Diff** is the difference in the objective values between the optimal and the feasible solution found with **DFS-feas** method,
- **Gap** is the difference between the feasible solution value and the global lower bound value in the branch-and-bound tree,
- **DecT** is the decrease in the computational time achieved by solving the problem to feasibility.

We observe from Table 6.6 that in the vast majority of test cases the gaps between the global lower bound value in the tree and the first found integer solution are zero. In all other cases the gaps are so low, that the differences between optimal and feasible integer solutions are very small, approximately 0.2%. We also observe that noticeable decreases in the computational times can be achieved by solving the problems to feasibility. We therefore suggest to terminate the branch-and-price algorithm as soon as the first feasible integer solution is achieved with the depth-first search of the tree. Alternatively, the branch-and-price can continue until the gap between the lower bound and the upper bound of the integer solution decreases to a certain value, e.g. 1%.

6.2.5 Test Conclusions

Based on the aforementioned test results we conclude that the multiple pricing strategy works better than the other implemented pricing strategies in the branch-and-price framework. We therefore suggest to implement the multiple pricing strategy as it is described in this thesis, i.e. by virtually partitioning all non-basic variables in subsets belonging to different train drivers. Then, at every pricing iteration, to return the minimum negative reduced cost column from every such subset, thereby adding several candidates to enter the basis. We have also observed that the way of generating the initial set of columns has an effect on the running time of the branch-and-price algorithm. The test results show that a fast convergence is achieved by using the limited subsequences strategy with the parameter $\eta_{\text{part}} = 15$ for generating the initial columns.

Another important observation confirms our expectation to the integer properties of the set partitioning based model to TDRP. Test results reported in this thesis as well as the preliminary tests described in Rezanova and Ryan [2006] and Rezanova and Ryan [2009] show that more than 95% of all generated TDRP instances are solved to optimality in the root node of the branch-and-price tree, i.e., by solving the LP-relaxation of the problem. Furthermore, we conclude that the objective function values of the first feasible integer solutions found by the depth-first search of the branch-and-price tree are very close to optimal solutions, and the gaps between the lower and the upper bounds are very small. We therefore conclude that the prema-

Table 6.6: Comparing optimal and feasible integer solutions.

ID	PrSt(η_{part})	App	#N	#It	#Var	Z^*	Diff	Gap	Time	DecT
S36	**MP**(5)	Opt	7	52	13152	2389			20.36	
S36	**MP**(5)	Feas	3	40	13150	2393	0.17%	0.17%	14.98	26.40%
S36	**MP**(10)	Opt	3	46	5470	2389			12.72	
S36	**MP**(10)	Feas	2	40	5470	2389	0.00%	0.00%	10.81	14.99%
S36	**FP**(15)	Opt	3	1138	3713	2389			133.47	
S36	**FP**(15)	Feas	2	1105	3713	2389	0.00%	0.00%	127.25	4.66%
S36	**SP**(15)	Opt	3	51	3587	2389			11.91	
S36	**SP**(15)	Feas	2	45	3587	2389	0.00%	0.00%	11.38	4.46%
S36	**MP**(15)	Opt	13	99	3878	2389			29.22	
S36	**MP**(15)	Feas	2	47	3697	2394	0.21%	0.21%	12.27	58.02%
S36	**MP**(20)	Opt	3	52	2840	2389			12.81	
S36	**MP**(20)	Feas	2	46	2840	2389	0.00%	0.00%	11.23	12.32%
S41	**PP**(15)	Opt	3	122	190	2360			3.20	
S41	**PP**(15)	Feas	2	105	190	2360	0.00%	0.00%	2.23	43.38%
S45	**MP**(1)	Opt	3	21	46489	2617			14.38	
S45	**MP**(1)	Feas	2	15	46489	2617	0.00%	0.00%	10.69	25.66%
S45	**SP**(15)	Opt	3	50	2246	2617			7.63	
S45	**SP**(15)	Feas	2	46	2246	2617	0.00%	0.00%	6.23	18.24%
S45	**MP**(20)	Opt	3	41	1908	2617			6.67	
S45	**MP**(20)	Feas	2	37	1908	2617	0.00%	0.00%	6.23	10.06%
S53	**MP**(10)	Opt	3	38	579	2393			2.53	
S53	**MP**(10)	Feas	2	32	579	2393	0.00%	0.00%	2.27	10.51%
S53	**MP**(15)	Opt	3	38	523	2393			2.44	
S53	**MP**(15)	Feas	2	35	523	2393	0.00%	0.00%	2.36	3.20%
S62	**SP**(15)	Opt	3	42	400	7122			2.13	
S62	**SP**(15)	Feas	2	37	400	7122	0.00%	0.00%	2.23	5.13%
S76	**MP**(1)	Opt	3	19	271424	9472			77.28	
S76	**MP**(1)	Feas	2	17	271424	9472	0.00%	0.00%	59.14	23.47%

ture termination of the branch-and-price algorithm should be implemented in TDR–DSS.

Table 6.7 presents solution details which are common for all pricing strategies. The following notation is used:

- $|K|$**(Init)** is the number of train drivers in the disruption neighbourhood. The number in parenthesis describes the initial value before disruption neighbourhood expansion,
- $|N|$**(Init)** is the number of train tasks in the disruption neighbourhood. The number in parenthesis describes the initial value before disruption neighbourhood expansion,
- **C** is the number of constraints in TDRP–LP,
- **Sh** is the number of train driver duties with a shifted check-out task, see Section 5.7,
- **Ext** is the number of train drivers with extended recovery period,
- **Add** is the number of added reserve drivers,
- Z^* is the objective function value of the solution,
- **Non** is the number of non-covered train tasks in the solution,
- **T** Computational times for solving all instances, except for S36 and S56, to optimality with the multiple pricing strategy **MP** to optimality. An early termination of branch-and-price is used to solve S36 and S56.

Solution results suggest that the disruption neighbourhood for all test instances with the recovery period of 1 hour and 1.5 hours was expanded by extending the recovery period for at least one train driver. We can therefore conclude that these recovery period lengths are too short to be applied to recover the train driver schedule, based on the tested data input. The expansion of disruption neighbourhood by adding reserve train drivers was necessary in all presented test cases. It was furthermore observed that at least 3 reserve train drivers were used in all recovery solutions. We therefore suggest to add all available reserve drivers to the initial disruption neighbourhood in further experiments. Since the test instances are generated independently of the previous recovery of the train driver schedule, the sizes of the instances grow as the start time of the recovery period increases. It

is also much harder to find a recovery solution that can cover all train tasks in the disruption neighbourhood, if no attempts to recover the schedule has been made. It it therefore important to start the recovery process of the train driver schedule as early as possible, and use all available information about the disruption. The computational times increase with the size of the disruption neighbourhood, but are still acceptable for employment of the presented solution method to solving TDRP in S-tog's daily operations.

Table 6.7: Solutions details for test instances.

ID	$\|K\|$(Init)	$\|N\|$(Init)	C	Sh	Ext	Add	Z^*	Non	T
S11	21 (12)	10 (8)	31	0	1	9	404	0	0.33
S12	21 (12)	14 (12)	35	0	1	9	404	0	0.41
S13	21 (12)	20 (20)	41	0	0	9	61	0	0.41
S14	23 (12)	28 (28)	51	0	0	11	61	0	0.56
S15	23 (12)	35 (35)	58	0	0	11	61	0	0.70
S16	24 (12)	41 (41)	65	0	0	12	76	0	0.94
S21	26 (17)	17 (13)	43	0	2	9	176	0	0.44
S22	26 (17)	23 (21)	49	0	1	9	176	0	0.50
S23	26 (17)	29 (29)	55	0	0	9	176	0	0.53
S24	28 (17)	38 (38)	66	0	0	11	176	0	0.67
S25	28 (17)	40 (40)	68	0	0	11	176	0	0.67
S26	29 (17)	46 (46)	75	0	0	12	174	0	0.91
S31	31 (22)	23 (11)	54	0	7	9	700	0	0.70
S32	35 (26)	37 (30)	72	0	4	9	1702	0	0.80
S33	46 (35)	56 (54)	102	0	1	11	1892	0	1.33
S34	56 (45)	81 (81)	137	0	0	11	1935	0	1.42
S35	67 (55)	120 (120)	187	0	0	12	2291	0	3.75
S36	72 (60)	151 (151)	223	0	0	12	2389	0	12.27
S41	39 (30)	33 (17)	72	3	8	9	2360	0	1.11
S42	43 (34)	44 (36)	87	4	3	9	2720	0	1.50
S43	55 (44)	62 (58)	117	4	1	11	2042	0	1.86
S44	68 (57)	98 (94)	166	4	1	11	2413	0	4.13
S45	76 (64)	131 (131)	207	4	0	12	2617	0	5.38
S46	81 (69)	169 (169)	250	4	0	12	2574	0	15.31
S51	38 (30)	25 (14)	63	7	5	8	2829	0	0.86
S52	48 (38)	38 (31)	86	7	4	10	2958	0	1.16
S53	57 (47)	63 (56)	120	7	3	10	2393	0	2.36
S54	67 (56)	88 (88)	155	7	0	11	2653	0	2.11
S55	72 (61)	118 (118)	190	7	0	11	2752	0	4.15
S56	81 (68)	160 (160)	241	7	0	13	2753	0	12.44
S61	48 (40)	32 (16)	80	9	8	8	7477	5	1.16
S62	59 (49)	53 (42)	112	9	6	10	7122	4	1.73
S63	69 (59)	78 (70)	147	9	3	10	6227	3	3.20
S64	77 (66)	101 (101)	178	9	0	11	6098	3	2.75
S65	82 (71)	133 (133)	215	9	0	11	5693	3	5.16
S66	88 (75)	165 (165)	253	9	0	13	5632	3	11.63
S71	54 (44)	43 (26)	97	7	10	10	10011	6	1.75
S72	62 (52)	60 (46)	122	7	7	10	9919	6	2.28
S73	71 (60)	80 (75)	151	7	3	11	10103	6	3.27
S74	74 (63)	101 (101)	175	7	0	11	9525	6	2.92
S75	81 (68)	132 (132)	213	7	0	13	9471	6	6.27
S76	85 (72)	166 (166)	251	7	0	13	9472	6	12.72

6.3 Rolling Time Horizon Cases

The recovery of the train driver schedule must continue during the day of operation until the order on the S-train network is restored. TDRP instances are therefore generated and solved with a rolling time horizon, as described in Section 3.2.3. We test the rolling horizon framework on the same data input used for generating test instances described in Section 6.1. The rolling horizon scenarios are, however, generated with slightly different assumptions. We assume that the system, and hence the train driver dispatcher has the information about *all* disruptions within the chosen recovery period, including expected delay propagations. To compare, the test instances described in Section 6.2.1 are generated by assuming that the system only has the information about delayed trains which run during the first 5 minutes after the recovery start time.

The initial disruption neighbourhood of every TDRP instance is generated by adding all train drivers who are assigned to the disrupted train tasks within the recovery period and all reserve drivers available during the recovery period. We start to recover the train driver schedule at 12:00 and continue to recover until 20:00, using recovery periods of 2, 2.5 and 3 hours, and a monitoring interval of 30 minutes. We measure the quality of recovery solutions achieved by solving the rolling horizon TDRP with the three chosen recovery periods. The main measurement criteria is the number of times the train driver duties within the disruption neighbourhood have been changed during the whole time horizon. This number corresponds to the number of times a train driver dispatcher has to inform train drivers about changes in the duty, if the solutions suggested by TDR–DSS were approved and implemented. Other measurements are the number of passengering tasks, the number of taxi rides, and the number of train driver changes from train tasks to train tasks on København H. These connections are given high costs, as described in Section 4.2.4. Tables 6.8, 6.9, and 6.10 present the test results using the following notation:

- **T_G** is the computational time in seconds for generating duty graphs and other data structure,
- **T_S** is the computational time in seconds for finding the optimal solution,

- **Pas** is the number of passengering tasks in the recovery solution,
- **Taxi** is the number of assigned taxi rides in the recovery solution,
- **KH** is the number of driver changes at København H,
- **Ch** is the number of train drivers who's original duties changed in the recovery solution,
- **%Ch** is the percentage of **Ch** among all train drivers in the disruption neighbourhood.

There was not need to expand the disruption neighbourhoods of any rolling horizon test instances during the whole time horizon, partly because the reserve train drivers were added to the initial disruption neighbourhood, and partly because all disruptions during the recovery period were available. There were no uncovered train tasks in the recovery solutions either. We have therefore omitted the details of **Sh**, **Ext**, **Add**, and **Non** from the tables. The rolling time horizon tests perform best with respect to the number of changed duties during the hole time horizon when the recovery period of 2 hours is applied. The train driver dispatcher would have to inform drivers 59 times compared to 70 and 87 times if the recovery periods of 2.5 and 3 hours are applied, respectively. The number of used taxi rides is highest if the shortest tested recovery period is applied, while the longest tested recovery period only generated 3 taxi rides. The taxi rides are very expensive in the solution and they are only used when no other possibility is available. It is therefore clear that the shortest recovery period required the largest number of taxis. The number of passengering tasks is, however, the lowest among the three tests, when the 2-hour recovery period is used. None of the solutions contain the unattractive train driver changes from train task to train task at KH.

Note that the first runs, R13_1, R14_1, and R15_1, contains the largest number of disrupted trains compared to the subsequent runs. In the subsequent runs only disruptions occurring between the end of the first recovery period and the end of the next recovery period are added to the problem, while the rest of the recovery period contains non-disrupted, i.e. solved in the previous runs, duties. This is particularly evident in the test case R15_1 with the longest recovery period. Hence, the subsequent problems in the rolling horizon framework are easier to solve. This is also evident from the number of

pricing iterations, the number of generated variables, and, as a consequence, from computational times, even though the differences are insignificant.

As we can see from the test runs, there were no more delays and re-routing of trains after approximately 18:30, and a very few train tasks still remained cancelled afterwards. According to the disruption records, the last cancelled train task was originally scheduled to depart at 19:24 from KH. It might seem strange that trains were still cancelled, even though there were no influential delays. There is, however, a natural explanation to that, since the train tasks were cancelled as a part of the aforementioned line cancellation recovery strategy on the S-train network. The re-insertion of the train lines back to operation is a non-trivial task (see Jespersen Groth et al. [2006]). Therefore, it usually takes some time from the decision about re-insertion of train lines to the actual implementation of the re-insertion. The order on the S-train network was completely restored after approximately 20:00. Zero objective function values of the solutions to the test instances R13_13, R14_12, and R15_11 confirm that.

The computational times are much smaller than the tests conducted in the previous section. All rolling time horizon instances were solved in the roon node of the branch-and-price tree. We can conclude from the above tests that the level of information is very important for how well the TDRP is solved. Compared to the test instances generated in Section 6.2.1 with a limited information about delays in the recovery period, the rolling horizon tests with full information obtain better solutions to the recovery. It is apparent that with the full information availability and a continues recovery, it is an easy optimization task to recover the train driver schedule when disruptions occur on the S-train network.

Table 6.8: Rolling time horizon test results with the recovery period of 2 hours.

ID	RecPer	Del	Turn	Can	T_G	N	It	V	C	K	N	Z^*	T_S	Pas	Taxi	KH	Ch	%Ch
R13_1	12:00–14:00	21	8	5	0.67	1	9	207	59	28	31	1563	0.34	6	3	0	11	39%
R13_2	12:30–14:30	5	5	17	0.20	1	10	170	49	23	26	1373	0.34	5	3	0	6	26%
R13_3	13:00–15:00	3	3	28	0.14	1	9	157	48	23	25	885	0.30	6	1	0	7	30%
R13_4	13:30–15:30	5	6	37	0.22	1	6	187	60	32	28	1100	0.25	9	1	0	7	22%
R13_5	14:00–16:00	7	7	41	0.27	1	6	160	53	30	23	1129	0.27	13	0	0	9	30%
R13_6	14:30–16:30	3	3	54	0.17	1	6	151	50	30	20	375	0.23	4	0	0	3	10%
R13_7	15:00–17:00	2	2	74	0.22	1	10	274	67	38	29	453	0.41	3	0	0	6	16%
R13_8	15:30–17:30	3	3	79	0.22	1	6	164	57	38	19	449	0.19	1	1	0	2	5%
R13_9	16:00–18:00	5	5	79	0.33	1	7	264	66	44	22	874	0.27	4	1	0	3	7%
R13_10	16:30–18:30	3	3	73	0.14	1	6	152	49	39	10	471	0.20	2	1	0	1	3%
R13_11	17:00–19:00	0	0	73	0.17	1	5	151	51	38	13	337	0.20	4	0	0	2	5%
R13_12	17:30–19:30	0	0	61	0.78	1	5	101	36	29	7	54	0.17	0	0	0	2	7%
R13_13	18:00–20:00	0	0	50	0.46	1	2	30	15	14	1	0	0.13	0	0	0	0	0%
														57	11	0	59	

Table 6.9: Rolling time horizon test results with the recovery period of 2.5 hours.

ID	RecPer	Del	Turn	Can	T_G	N	It	V	C	K	N	Z^*	T_S	Pas	Taxi	KH	Ch	%Ch
R14_1	12:00–14:30	26	13	18	1.70	1	14	496	100	40	60	1766	0.64	9	2	0	24	60%
R14_2	12:30–15:00	3	3	28	0.22	1	10	214	56	25	31	880	0.22	5	1	0	7	28%
R14_3	13:00–15:30	5	6	37	0.30	1	8	292	74	32	42	912	0.22	10	0	0	9	28%
R14_4	13:30–16:00	7	7	41	0.39	1	8	283	74	33	41	1475	0.20	13	1	0	12	36%
R14_5	14:00–16:30	3	3	58	0.25	1	7	200	65	32	33	911	0.23	3	2	0	3	9%
R14_6	14:30–17:00	2	2	77	0.28	1	8	293	78	39	39	697	0.25	8	0	0	6	15%
R14_7	15:00–17:30	3	3	91	0.36	1	8	226	66	39	27	449	0.25	1	1	0	2	5%
R14_8	15:30–18:00	5	5	89	0.56	1	10	397	76	43	33	874	0.33	7	1	0	3	7%
R14_9	16:00–18:30	3	3	88	0.28	1	6	168	54	41	13	471	0.17	2	1	0	1	2%
R14_10	16:30–19:00	0	0	87	0.27	1	6	211	62	43	19	482	0.19	5	0	0	2	5%
R14_11	17:00–19:30	0	0	81	0.78	1	4	133	44	36	8	82	0.93	1	0	0	1	3%
R14_12	17:30–20:00	0	0	65	0.62	1	1	30	15	15	0	0	0.78	0	0	0	0	0%
														64	9	0	70	

Table 6.10: Rolling time horizon test results with recovery period of 3 hours.

ID	RecPer	Del	Turn	Can	T_G	N	It	V	C	K	N	Z^*	T_S	Pas	Taxi	KH	Ch	%Ch
R15_1	12:00–15:00	29	16	39	3.93	1	18	861	132	47	85	1880	1.61	16	0	0	32	68%
R15_2	12:30–15:30	5	6	37	0.58	1	10	431	84	31	53	743	0.42	8	0	0	8	26%
R15_3	13:00–16:00	7	7	41	0.75	1	12	536	95	34	61	1107	0.75	11	0	0	14	41%
R15_4	13:30–16:30	3	3	58	0.47	1	10	399	84	33	51	1085	0.55	4	2	0	7	21%
R15_5	14:00–17:00	2	2	81	0.67	1	11	548	88	37	51	781	0.59	8	0	0	10	27%
R15_6	14:30–17:30	3	3	94	0.52	1	9	411	81	37	44	509	0.47	5	0	0	7	19%
R15_7	15:00–18:00	5	5	101	0.86	1	13	520	92	42	50	566	0.73	6	0	0	5	12%
R15_8	15:30–18:30	3	3	98	0.45	1	7	216	59	38	21	471	0.28	2	1	0	1	3%
R15_9	16:00–19:00	0	0	102	0.39	1	4	189	60	40	20	427	0.16	5	0	0	2	5%
R15_10	16:30–19:30	0	0	95	0.13	1	6	105	43	36	7	33	0.19	0	0	0	1	3%
R15_11	17:00–20:00	0	0	85	0.13	1	4	83	30	24	6	0	0.17	0	0	0	0	0%
														65	3	0	87	

Chapter 7

Future Research and Conclusion

7.1 Future Research

In this section we describe a few suggestions for the future research and development of the train driver recovery problem decision support system. They concern alternative modelling and solution approaches, general algorithmic improvements of the programming code and implementation details.

7.1.1 Alternative Objective of Train Driver Recovery

Train driver dispatchers at S-tog aim at disturbing the original duties of as few train drivers as possible in recovery solutions. We therefore suggest an alternative formulation to express this. Let y^k be a binary variable that is equal to 1 if the driver $k \in K$ is included in the recovery solution, and 0 otherwise. The cost c^k express the unattractiveness of including a particular driver k to the recovery solution. Let z_i^k be a binary variable that is equal

to 1 if driver k covers train $i \in N$, and 0 otherwise. Then a formulation of a train driver recovery problem which aims at minimizing the number of train drivers included in the recovery solution is the following:

$$\text{(TDRP-min)} \quad \text{Minimize} \quad \sum_{k \in K} c^k \, y^k \tag{7.1}$$

$$\text{Subject to} \quad \sum_{k \in K} z_i^k = 1 \qquad \forall i \in N, \tag{7.2}$$

$$z_i^k - y^k \leq 0 \qquad \forall k \in K, \, \forall i \in N, \tag{7.3}$$

$$y^k \in \{1, 0\} \quad \forall k \in K. \tag{7.4}$$

$$z_r^k \in \{1, 0\} \quad \forall k \in K, \, \forall i \in N. \tag{7.5}$$

The set of constraints (7.2) ensures that every train task i in the disruption neighbourhood is covered by a train driver in K. The linking constraints (7.3) ensure that if driver k covers train task i then the train driver variable $y^k = 1$ in the solution. Since the objective function (7.1) minimizes the number of train drivers in the solution, then variable $y^k = 0$ if $z_i^k = 0$. In order to ensure feasibility of the solution to TDRP-min, an artificial variable $s_i, i \in N$ can be added to every partitioning constraint (7.2), as well as to be added to the objective function with a large cost. Artificial variables play the role of cancellation decision variables, similar to the artificial variables $f_i, i \in N$ in the formulation of TDRP–LP. The problem formulation of TDRP-min does, however, not have the same integer properties as TDRP, since there is no GUB constraints present in the latter formulation. It means that TDRP-min is not as "easy" to solve as the set partitioning problem with GUB constraints by solving its LP-relaxation.

7.1.2 Integrated Approach

The most interesting and important research direction is towards the integrated recovery of the railway operations. In the optimization model and the solution method implemented in this project the hierarchical approach to recovery is assumed. The train driver schedule is recovered *given* the timetable and the rolling stock schedule. If a particular train task i cannot be covered

by the drivers within the expanded disruption neighbourhood of a particular recovery solution, the network traffic control center must take a decision if the departure of the train task i is to be delayed, re-routed or cancelled.

Ideally, the disruption management decision support should include delaying, re-routing and cancellation possibilities of all train tasks within the disruption neighbourhood. Furthermore, feasibility of these decisions cannot be ensured unless the rolling stock schedule is taken into considerations. It is therefore a difficult and a challenging task to develop an integrated railway recovery tool. The integrated railway recovery approaches have not yet appeared in the operations research literature. Potthoff et al. [2008] include cancellation decision variables in the crew re-scheduling model. These variables, however, serve exactly the same purpose as the artificial variables $f_i, \forall i \in N$ in the TDRP-LP model presented in Section 5.3, namely to detect which train tasks cannot be covered by the train drivers within the core problem. The model does not ensure that the rolling stock schedule does not become infeasible when a certain train task is cancelled in the solution. The integrated re-timing of departures and crew recovery is implemented in a very few applications of the airline crew recovery (Abdelghany et al. [2004]). There are also some integrated solutions within the airline scheduling (Mercier and Soumis [2007], Weide et al. [2009]). Solving integrated problems to optimality is computationally expensive, even with decomposition approaches. Weide et al. [2009] present an alternative approach, where the crew scheduling and the aircraft scheduling problems are solved iteratively. A similar approach seems to be suitable for recovery applications, also within railway disruption management.

7.1.3 General Strategy for Initial Disruption Neighbourhood

We have concluded that the recovery periods of approximately 2–3 hours are suitable for recovering the train driver schedule based on the tested disruption data. Initial disruption neighbourhoods are generating by only including the disrupted and the reserve train drivers. However, other recovery period lengths and other strategies for including train drivers can possibly work

better in other disrupted situations. An important issue for further research is to determine a general strategy for adjusting the recovery period length and other parameters to different types of disruptions. Guo [2005a] (see Appendix A) consider the airline crew recovery problem and present a *strategy mapping* approach to choosing the solution method and other parameters, such as the length of the recovery period. These ideas can be used to develop a general strategy for generating disruption neighbourhoods according to the magnitude and the location of disruptions, and considering the recovery initiatives towards the timetable and the rolling stock recovery on the S-train network, and other factors.

7.1.4 Exploration of Disruption Neighbourhood

The expansion of the disruption neighbourhood is another topic which must be explored further. Adding more train drivers to the disruption neighbourhood can improve the quality of the solution. We have partly implemented the possibility of adding train drivers who are available to cover the non-covered train tasks, but the strategy needs further refinement. Potthoff et al. [2008] present a detailed strategy for adding new train drivers to the disruption neighbourhood. These ideas can be used as an inspiration for further research within this area. An open question in the area of the disruption neighbourhood expansion is how to ensure that the recovery solution restricted by the chosen disruption neighbourhood is as close to the *global* optimal solution given the information available about the disruption, and how to find the answer without having to include all train drivers in schedule and considering a recovery period that lasts for the rest of the day?

7.1.5 Refining the Limited Subsequences Strategy

One way to refine the limited subsequences pricing strategy is to calculate a value of the filter η_v^k for every vertex v in the duty graph G^k. Then $\Delta\eta_v^k$ is also calculated individually for every vertex v:

$$\Delta\eta_v^k = \lceil \frac{|\delta^+(v)|}{\eta_{\text{part}}} \rceil. \tag{7.6}$$

The more precise value of η_v^k makes it easier to control the speed of the deepening of the limited subsequences though the the value of η_{part}. The disadvantage of this refinement is in increasing number of calculations necessary to perform at every major iteration of the pricing strategy.

Another refinement of the limited subsequences strategy is to make it possible to control which columns are collected at each pricing iteration by using the knowledge about the practical problem for sorting the outgoing arcs of every vertex in a duty graph. It is not always *the cheapest* arcs that need to be considered first, particulary, if a feasible recovery solution is difficult to achieve. Consider a situation where the only way a feasible recovery duty can be generated for a particular train driver k is to include a deadheading in a taxi after the completion of a certain train task represented by vertex v. Since the taxi arcs have high costs, and the outgoing arcs of v are sorted in the increasing order of their costs, the particular taxi arc is considered after the limited subsequences have deepened enough to reach the arc. If the knowledge about the necessity of this particular taxi arc was available, this arc would be given a lower cost, and the infeasibility regarding the train driver k would have been resolved much sooner in the column generation process. A more refined limited subsequences strategy can also allow to force particular train tasks to be included in the recovery duty of a particular train driver at the beginning of the column generation by changing the cost of arcs.

7.2 Conclusion

An important operational problem is considered in this thesis, taking a starting point in the operations of a Danish passenger railway company DSB S-tog A/S. The train driver recovery problem (TDRP) occurs during the day of operation, when the daily train driver schedule becomes infeasible due to disruptions on the railway network. The problem is closely related to two important areas within operations research applications: crew planning problems in the railway industry and disruption management with focus on crew recovery in public transportation. The operations research publications within these areas have been thoroughly reviewed in order to obtain knowledge within the state-of-the-art on modelling, solution approaches and implementations.

We give a general introduction and references to operations research related work at different stages of the planning process in the passenger railway industry. We have also explored the practical aspects of the train driver scheduling and dispatching at S-tog. During the project, a close contact to the planners and operations researchers at the Production Planning Department at S-tog has been established, and we have taken several opportunities to observe the operations of the network control center and train driver dispatchers. A detailed description of the tactical and operational crew planning processes at S-tog is presented, and an insight into the timetable and rolling stock dispatching on the S-train network is given.

We have developed a prototype for a train driver recovery decision support system for the train driver dispatchers at S-tog. The prototype is programmed in C#.NET. The decision support system solves TDRP instances with a rolling time horizon. In order to generate a TDRP instance we need a timetable, a train driver schedule, and a list of disruptions to be applied to the timetable and the train driver schedule. We consider the following types of disruptions affecting the timetable: departure and arrival delays, train re-routings by turning the train before it reaches the terminal station, cancellations of single train tasks, and cancellations of train lines which result in cancelling all train tasks of these lines for a period of time. We also consider disruptions that affect the train driver schedule directly, and only have an indirect effect on the timetable. These are the train driver absences

from duties due to e.g. acute sicknesses and the train driver delays for other reasons than the disrupted timetable. For every TDRP instance we choose a certain recovery period and a certain recovery start time. The time difference between recovery start times of two subsequent TDRP instances is called a monitoring interval. During the monitoring interval new information about disruptions become available to the system either directly from the network traffic control center or from the train driver dispatcher. By solving TDRP instances with a rolling horizon, the train driver schedule is constantly recovered until the operations on the S-train network are back to the normal state.

The problem space of every TDRP instance is limited by a certain disruption neighbourhood, which contains a set of train drivers and a set of train tasks that are to be assigned to the drivers. Since the main objective of the recovery solution is to make as few modification to the original train driver schedule as possible, the size of the disruption neighbourhood is restricted by only including the train drivers disrupted within a certain recovery period. If the size of the disruption neighbourhood is not sufficient for finding a recovery solution, we expand the disruption neighbourhood by extending the recovery period and adding other train drivers. Computational experiments show that it is appropriate to include all available reserve drivers to the initial disruption neighbourhood. The rolling time horizon recovery tests show that it is possible to recover the train driver schedule with the available reserve.

Different solution approaches to solving the train driver recovery problem have been considered. We formulate TDRP as a set partitioning problem, where variables (columns) represent recovery duties of the train drivers. The problem contains two sets of constraints. The generalized upper bound (GUB) constraints ensure that every train driver is assigned to exactly one recovery duty, while the set partitioning constraints ensure that every train task is assigned to exactly one train driver. Due to the presence of the GUB constraints the model possesses strong integer properties: every constraint submatrix with the GUB constraint covering a particular train driver is perfect. The integer properties force the LP-relaxation of TDRP to have optimal integral solutions in the vast majority of the test instances. Computational experiments show that approximately 96% of all generated TDRP test instances are solved to optimality in the root node of the branch-and-price tree.

The perfect structure of the driver submatrices also ensures that the fractions in TDRP–LP can only occur when two or more train drivers compete for covering the same train task. This observation suggests an efficient way to search for integer solutions, based on constraint branching. Computational experiments show that in cases where branching is required, the values of the first found feasible integer solutions have approximately 0.2% gap from the best lower bound value. The costs of recovery duties in the objective function are not represented by the real costs of duties. Artificial cost penalties are applied to those recovery duties, which are unattractive from the point of view of minimizing modifications to the original duties. We can therefore conclude that it is sufficient to solve TDRP by finding the first integer solution, and the branch-and-price algorithm can terminate before the optimal solution is found.

We use a limited subsequences strategy to generate the initial set of columns. The idea of the limited subsequences is to limit the number of train tasks a driver can perform after completion of every task in the duty. Since the train task subsequences are expressed by the arcs in duty graphs, we only allow to consider a limited number of outgoing arcs of every vertex on the duty graph when generating recovery duties with the limited subsequences strategy. This strategy, if carefully tuned, is efficient for generating a good initial set of columns for TDRP–LP. We implement several pricing strategies. Computational experiments show that the multiple pricing strategy outperforms the other strategies. With the multiple pricing strategy, several columns are added to the restricted master problem at each iteration. Each column corresponds to the minimum negative reduced cost recovery duty for one train driver in the disruption neighbourhood. After every pricing step the simplex algorithm performs several iterations, and more than one generated column can therefore enter the basis every time the restricted master problem is solved.

During this project we have considered different data input provided by S-tog. The preliminary results reported in Rezanova and Ryan [2006] employ the train driver schedule from year 2005 and simulated disruptions by cancelling one train line. Afterwards, the train driver schedule from year 2007 was adopted and the real-life timetable disruptions applied. The results are reported in Rezanova and Ryan [2009]. In this thesis we report the computational experiments with the algorithmically improved version of the program,

using the same real-life data input. We consider one day of operations from January 2007, where due to a broken railroad switch the timetable and the train driver schedule were severely disrupted from 12:00 to 20:00. As a consequence of the disruption, several train lines were cancelled and many trains re-routed. We generate independent test scenarios with disruption neighbourhoods of 21–88 train drivers and 10–169 train tasks. All solutions are found within 16 seconds, using the multiple pricing strategy with the best parameter for collecting initial set of columns and applying the early termination of the branch-and-price algorithm. Computational experiments show that recovery periods of 2–3 hours are most suitable for the recovery. The chosen solution strategy is applied to the rolling time horizon test scenarios, where TDRP instances are solved with recovery periods of 2, 2.5, and 3 hours, using the 30 minutes monitoring interval. The largest size of disruption neighbourhood generated with the rolling horizon contains 47 train drivers and 85 train tasks. The largest rolling horizon instance is resolved within 2 seconds, while optimal solutions to the vast majority of instances are found within less than 0.5 seconds. Based on the computational experiments conducted during this project we can conclude that the solution method employed in the prototype for the train driver recovery decision support system is suitable for implementing in the real-time environment.

The feasibility and the quality of generated recovery solutions have been continuously evaluated by train driver dispatchers at S-tog. For the purpose of presenting recovery solutions we have developed a graphical user interface (GUI) of the prototype. The GUI was unexpectedly appreciated by the planners and dispatchers. This shows that operations research applications must not only focus on the optimization methods, but also on providing a comprehensible link to the end-users of the systems under development. Currently, S-tog is working on different scenarios for implementing the prototype developed during this thesis into the operational environment. We therefore conclude that this thesis contributes not only to operations research applications from the scientific point of view, but also to a potential improvement of daily operations of DSB S-tog A/S, which can result in a higher service level provided for the passengers on the S-train network.

Appendix A

Airline Crew Recovery Review

To our knowledge, Johnson et al. [1994] is the first publication regarding the airline crew recovery. The problem is formulated as a set covering problem with additional constraints. The authors consider recovering pilot pairings when a single flight is delayed at a single airport. An approach for identifying crew to be involved in the recovery solution is proposed. Experiments are conducted based on data files supplied by Northwest Airlines. All pairings for the crew recovery problem are generated a priori from a time-line network and the set covering problem is solved using MINTO (Nemhauser et al. [1994]). Three small test scenarios are described, but the running times are not presented.

Teodorović and Stojković [1995] suggest a model to reduce the airline schedule disturbances, where the main focus is on recovering aircraft from disruptions. Recovery crew rotations are scheduled using a first in first out (FIFO) principle and a sequential dynamic programming approach. In the FIFO algorithm the first arriving flight leg at an airport is linked to the first departing leg. The rotation finishes when the chain of flight legs cannot be extended due to the crew working hours, number of take-offs, breaks etc. It is not taken into account that the crew has to end the rotation at a certain airport. In the sequential approach, a crew rotation is generated for the "first crew", then all flight legs used in the rotation are excluded from the schedule, and a recovery rotation is generated for the next "first crew". The "first crew" is suggested to be the crew which is able to fly the shortest path

route of legs, where the length of the path is determined by the time the crew spends on the ground after finishing each leg in the path. Both methods are far from optimal, but the authors ague that it is "easy" to deadhead crew, if it arrives at a different airport than planned.

Wei et al. [1997] is the first published work exclusively dedicated to the airline crew recovery. A very similar publication is reported by the same authors in Song et al. [1998]. The problem is modelled as an integer multi-commodity network flow problem on a time-space network, where vertices represent flight legs and arcs represent feasible crew transitions between flights. Each crew is required to be back to its original schedule at the end of the recovery time window, which is represented by return vertices in the network. A crew represents a commodity in the network and a directed path through the network ending up in the crew return vertex represents a feasible pairing. The set covering problem formulation of the crew recovery problem is presented. A heuristic-based search algorithm is proposed in order to repair the broken crew pairings. The solution process is represented by a search tree, where each node represent a set of uncovered flights and a list of pairings modified so far in the search process. At each node, an uncovered flight is picked and a candidate crew list is generated to cover this flight. Different crew candidates lead to different branches of the solution tree. All generated pairings must be legal, return to the designated return vertex and be as close to the original paring as possible. If all flights are covered, the node represent a feasible solution. The preprocessing step to the algorithm is employed in order to try to return every affected crew back to its original pairing. A negative-cost shortest path algorithm is employed to find a feasible path in the network, which represent a pairing as close to the original as possible. Flights which are still uncovered and the modified parings after the preprocessing step form the first node of the solution tree. A few computational experiences are given to demonstrate the application of the algorithm. The algorithm terminates when a requested number of solution is found or when a certain time limit is achieved. For the first set of test instances, the flight schedule of 18 flight legs covered by 6 pairings and 1 reserve crew is used. In 6 proposed disruption scenarios one or two flights are delayed or cancelled, resulting in modification of two or three pairings. Finding the fist feasible solution is achieved within 0,72 to 6,50 seconds on HP 715/100. The optimal solution is found by requesting to find all possible solution nodes, which is achieved within 0,72 to 24,22 seconds (8 seconds on average). The second

set of test scenarios was generated for 51 flights and 18 pairings, cancelling or delaying one or two flights. Requesting the termination of the algorithm after 3 solutions are found, solutions for the 8 test cases are found within 0,29 to 5,76 seconds on HP9000/K420 parallel system.

The framework for a decision support system presented in Wei et al. [1997] was further developed by CALEB Technologies (www.calebtech.com) and implemented in Continental Airlines, as reported Yu et al. [2003]. The success of the implementation helped the airline to quickly recover from major snowstorms and 9/11 terrorist attack in 2001. The project was awarded the 2002 Franz Edelman Award for Achievement in Operations Research and the Management Sciences. The decision support system incorporates an optimization server with the system operations control database and other data systems. The solver generates up to three solutions in order to give the operator a flexibility to choose the most appropriate decision. Real-life test scenarios are reported, run on one processor of a Sun system with four 300-Mhz UltraSPARC II processors. Test problems with up to 20 affected flights were resolved within 0,97-58,89 seconds on average, while crew recovery problems, where between 21 and 40 flights were affected, took approximately 200-300 seconds on average to resolve.

Stojković et al. [1998] consider an operational airline crew scheduling problem, where individual work schedules for the crew have to be modified during an operational phase. The problem is formulated as a set partitioning problem which is aimed at minimizing the cost of modified pairings within a certain operational period. Decision variables represent legal pairings, which are generated from space-time oriented graphs as resource constrained paths, where resources represent local pairing constraints, e.g. time away from base. The problem is solved with branch-and-price, where the linear programming relaxation of the problem is solved with column generation, using the GENCOL 3.0 optimizer, which was successfully employed to solve the airline crew pairing problem Desaulniers et al. [1997], among others. Four test scenarios from an unnamed U.S. carrier pairings are generated, considering three delayed flights and one indisposed crew member. Solutions to the two test scenarios defined over one operational period (1 day) was found within 7,21 and 30,42 seconds on HP 9000/715. Solutions to the two remaining scenarios defined over 7 days of operations were found within approximately 4 and 20 minutes.

Stojković and Soumis [2001] extend the crew recovery problem of Stojković et al. [1998] with a possibility to delay scheduled flights explicitly through the problem formulation. Some flights have fixed departure times, some others have more flexible times in terms of a flight specific time window. The problem is formulated as a multicommodity network flow with additional constraints, and is solved using column generation with a master problem and a subproblem per pilot. The solution may include the use of reserve pilots, treated as extra artificial commodities in the problem. The model and solution method has been tested on three problems. The largest problem has 59 pilots and 190 flights, of which 52 are originally delayed. All problems are tested with and without reserve pilots, allowing delays of flights and with a fixed flight schedule. The results are encouraging, both in terms of quality and in terms of computing times. Stojković and Soumis [2005] builds on the model derived in Stojković and Soumis [2001], but extends it to work with multiple crew members, which makes the situation addressed more realistic. The extension is achieved by using a number of copies of each flight corresponding to the number of crew required. A set of constraints ensure that the departure times for all copies of each flight are added to the model. The solution process is similar to that described in Stojković and Soumis [2001]. Three different models are tested: One corresponding to that from the previous work with strict flight covering constraints, one in which there is a linear cost for missing crew members, and one with a cost for each flight with missing crew. It is demonstrated that using both the second and the third model, substantial improvements compared to the initial situation can be obtained. However, the solution times experienced for large problems are prohibitive in an on-line situation (more than an hour).

Lettovsky et al. [2000] formulate the crew recovery problem as a generalized set covering problem, where rows represent flight segments that need to be covered and columns represent feasible pairings generated from the flight segments. It is also possible to assign no pairing to a crew and to cancel a flight segment. Flight segments are generated in the preprocessing step and are sequences of flight legs without crew swapping opportunities. The set of flight segments includes all uncovered flights and flights belonging to the crews which are chosen for the recovery in the preprocessing step. An efficient tree-based data structure is used for storing duties and pairings, which decreases memory requirements and allows fast computation of reduced costs. The linear programming relaxation of the crew recovery model is solved with

primal-dual subproblem simplex method due to Hu and Johnson [1999]. A branch-and-bound procedure is used to find a good integral solution. The branching procedure contains a sequence of three steps. First, fractional cancellation variables in the model are resolved. Second, the fractional deadheading variables are resolved. Third, branching on follow-ons (also known as constraint branching due to Ryan and Falkner [1988]) is performed. Three test scenarios from an unnamed U.S. carrier's data are generated with 1296 flight legs and 177 pairings in the original schedule. In the first scenario 3 flights are delayed due to maintenance-related disruption. The second scenario involves 11 delayed flight arrivals and 1 cancellation due to a thunderstorm. The third scenario involves 23 delayed flights and 12 cancellations in three airports hit by a moving snowstorm. Each scenario is solved with three levels of restrictions. At the first level the flight segments are not allowed to be disconnected and only 3 potential crews are included for swaps for each missed connection. Solutions to the first restriction level are found within 1-6 seconds on a PC/Pentium 150 MHz, 48 MB RAM, and 7 and 21 flights were not covered in the second and third scenario, respectively. When solving the recovery problems with allowing flights within flight segments to be disconnected and adding 10 potential crews for swapping, the number of uncovered flights in the solution was reduced to 2 and 6 in the second and the third scenario, respectively, but the computational time increased to 6, 115 and 97 seconds for the three scenarios.

Medard and Sawhney [2007] (an earlier version in Medard and Sawhney [2004]) point out that the crew recovery models published so far generate crew pairings for the whole crew, without incorporating individual rostering information of each crew member. The generated recovery solutions can potentially be illegal from rostering point of view. The authors present a crew recovery model, where rostering constraints are incorporated into the flight-based set covering crew recovery model alone with the pairing constraints. An acyclic activity network is build for each crew member within a recovery window. Carry-in and a carry-out nodes are defined individually for each crew member, representing the beginning and the end of the recovery window, respectively. Legal paths through the activity network represent all potentially legal rosters between the carry-in and the carry-out for a specific crew member. Two solution methods are proposed. A greedy enumeration approach involves a depth-first collection of legal recovery rosters for each crew member and then solving the set covering problem, selecting a roster

for each crew member. The second approach employs the crew pairing column generation method developed in Carmen Systems (Hjorring and Hansen [1999]) and the integer programming solver due to Wedelin [1995]. Test instances are generated from unnamed data within a recovery time window of 48 hours, where between 14 and 885 crew members are captured by the recovery window. The disruptions involve up to 251 uncovered flights and up to 77 illegal crew rosters. The depth-first search approach for generating rosters generally works faster than the column generation approach due to the large overhead for setting up the duty network in the latter case, and the authors conclude that the column generation framework has to be refined. Single base test problems are resolved within 15-76 seconds with depth-first search method, while the column generation approach required between 27 and 82 seconds on a Linux laptop (256 K RAM, 1 GHz CPU). Multi-base test problems are resolved within 12-584 and 14-840 seconds with the two approaches, respectively.

Abdelghany et al. [2004] present a decision support tool for recovering trippairs of a hub-and-spoke airline operation. A trippair is a sequence of two duties for one crew with a rest period in between. The recovery horizon is divided into a set of consecutive stages. The crew recovery problem is solved at each recovery stage. Several preprocessing steps are applied, including shifting the problem occurrences from the spokes to hubs, adding undisturbed crew to the recovery for covering open flights, grouping flights into resource-independent sets, where flights within the same set cannot be covered by the same crew and defining a set of candidate crew members to cover each flight within the recovery period and the cost of each candidate-flight pair. The problem of assignment of crew members to flights is formulated as a mixed integer program, where linear variables for flight departure times allow to minimize the total flight delay in the objective function, while the assignment variables take care of the minimum cost crew assignment to the flights. The problem is solved using using CPLEX Callable Library solver (www.ilog.com). The recovery solution is only applied to the open flights in the current recovery stage, leaving other assignment decisions to the later recovery stages. A disruption scenario from the operations of a major U.S. airline is used as a test case, were 18 crew members were disrupted. After the preprocessing steps, 121 crew members were collected for the recovery to cover an unmentioned number of flights. The recovery problem was solved in 1 minute and 51 seconds on HP B.11.00 U 9000/800 system.

Guo [2005a] present a decision support framework for recovering airline crew rosters. The crew recovery problem is formulated as a set partitioning problem, aimed at minimizing the modifications from the planned schedule. Two solution methods are implemented, a standard column generation with LP relaxation of the set partitioning problem and a heuristic method based on a hybrid of a genetic algorithm with a local search. The solution strategy is chosen at a preprocessing step using the Analytic Hierarchy Process (due to Saaty [1980]) by a *strategy mapping*, which is the main focus of the article. The strategy mapping provides a method to prioritize alternative solution methods for solving the crew recovery problem by evaluating a set of criteria, such as additional cost for recovering the schedule, solution time, the number of crew members that need to be notified, the period of time starting from the first updated flight to the last one, and the number of disturbances to crew. The authors define following factors which describe the complexity of the disruption: the number of delayed flights, cancelled flights, new flights to be operated, available crew members, and daily flights in average), a strategy for solving the crew problem is chosen. A strategy is a combination of solution methods (column generation and genetic algorithm) and relevant parameters, like recovery period length. A case study containing data from a European airline with several home bases is presented to demonstrate the strategy mapping process. The disruption involves 2 delayed flights, 1 cancelled flight, 2 new flights, 188 crew members and 85 daily flights on average during a 5 days recovery period. The author presents the way to compare three chosen criteria in order to choose between two solution method strategies, and conclude that in the presented case study the genetic algorithm solution method is preferred to the column generation method, and could produce an acceptable solution within approximately 3 minutes, which is a "dramatic reduction" compared to the column generation solution time. The work is based on the Ph.D. thesis by Guo [2005b].

Nissen and Haase [2006] formulate the crew recovery problem on a duty-period-based network. A duty period is a time when a crew is assigned to one or several flights between two rest periods. Instead of using a time-space network, where arcs represent flight legs or connections between flights, the authors build a network, where arcs represent all possible duty periods and rest periods for each crew within a chosen recovery period. Nodes in the network represent an airport at a specific time as well as the dummy crew nodes representing the beginning and the end of the recovery period. A resource

constrained path in the duty-period network, where resources determine the rest time and workload rules for the crew, represent a feasible sequence of duty periods, i.e. a part of the crew roster. The crew recovery problem is formulated as a network flow problem. A Dantzig-Wolfe decomposition is applied to the original model and the set covering master problem is solved with column generation and LP relaxation (using CPLEX 7.0 to solve linear problems). Integer solutions are found in a branch-and-price framework, where a branch-on-follow-ons (constraint branching) is applied. Artificial short-haul and medium-haul crew schedules for one fleet were generated based on a flight schedule of a European airline, since the original crew schedules were not available. The schedules included 450 and 927 flights in a week, respectively for the two schedules, while the number of crew to cover the flight schedule was not mentioned. Several artificial disruption test instances were generated, containing delaying, cancelling and adding new flights to the schedules. A recovery period of 48 and 60 hours was found to be the best (as a compromise between solution quality and solution time) for the short-haul and the medium-haul schedules, respectively. The authors conclude that particularly for the larger instances it could pay off to start solving the LP with a set of heuristically generated columns instead of a set of artificial variables only, that the largest overhead time is spent in the subproblem, and that only a very few test instances produced fractional solutions after solving the LP relaxation. Computational times for the tests run on a PC with an Atlon 1200 MHz CPU and 512 MB RAM are acceptable for operational environment for the best choice of model parameters.

Zhao et al. [2007] use grey programming to model the airline crew rescheduling problem. Grey programming is a part of the grey system theory proposed by Deng [1982], where a system is called "white" when the system information is fully known, "black" when the system information is unknown, and "grey" when the system information is partially known. The authors consider crew ready times, and arrival and departure times of the flights during irregular operations as interval grey variables $\otimes x$ with known lower and upper bounds, but with an unknown distribution information. The crew recovery problem is modelled as a grey programming model with binary variables (assignment of crew to flights) and grey variables (flight departure times variables). The model is a copy of the mixed integer programming model proposed by Abdelghany et al. [2004], with the only difference is that linear decision variables for departure and arrival times of flights and the linear parameter defining

the ready time for crew in the model of Abdelghany et al. [2004] are defined as grey variables in the model of Zhao et al. [2007]. Furthermore, the literature review in Zhao et al. [2007] is an exact copy of the literature review in Abdelghany et al. [2004]. A local search heuristic is applied to solve the problem. The authors present two solutions to one test case. However, neither the test case nor the computational details are presented in the paper.

Castro and Oliveira [2007a] and Castro and Oliveira [2007b] in almost identical publications present an implementation of the Distributed Multi-Agent System (MAS) to represent the operations control center of an airline company. The MAS includes a crew recovery agent, an aircraft recovery agent and a passenger recovery agent. The paper is focused on the architecture and test experiments of the crew recovery agent. A monitoring agent class of the crew recovery agent is responsible for monitoring crew events (e.g. non-assignments for some flights) and reporting to the crew finder agent class. The crew finder collects a list of solutions to the problem from the algorithmic agent classes and chooses the cheapest one using the crew payroll information. The authors do not mention what kind of algorithms and heuristics are used in the algorithmic agent classes to find solutions to the recovery problem. One test scenario is reported, where 15 crew members with different ranks are set to be absent from their duties at the same base. The proposed MAS method was compared to the human operator, where solutions to the test problem were compared by solution time and the cost of the solution (crew payment). The MAS recovery agent came up with a cheaper solution in 25 seconds compared to 101 seconds used by the operator.

Appendix B

Timetable Data Representation

Table B.1: The stopping pattern of the train nr. 10100 of line **A**.

*10100	1234567()	
UND		23:49
IH	23:51&	23:52
VLB	23:54&	23:54&
BSA	23:56&	23:56&
AVØ	23:59	23:59
FRH	00:01	00:01
ÅM	00:03	00:03
ELB		00:04&
NEL	00:05&	00:05&
SJÆ	00:06&	00:07
SYV	00:08&	00:08&
DBT	00:11&	0:11&
KH	00:14	00:15
VPT	00:16	00:16&
KN	00:18	00:18&
KK	00:21	00:21&
NHT	00:23&	00:23&
SAM	00:25	00:25&
HL	00:28	00:28&
BFT		00:30
GJ		00:31
JÆT		00:32
LY	00:33	00:33&
SFT		00:35&
VIR		00:37
HOT	00:38	00:38&
BI	00:42	00:42&
LI	00:47	00:47&
HI	00:54	

Appendix C

Train Driver Duty Data Representations

Table C.1: Train driver duty representation, type I.

DutyNr	TrainNr	Task Code	StartTime	EndTime	DepStation	ArrStation
209	00000	CIN	11:53	12:08	KH	KH
209	00000	BEV	12:08	12:12	KH	KH
209	50136	PIT	12:12	12:13	KH	KH
209	50136	TFF	12:13	12:52	KH	FM
209	00000	SPD	12:52	12:56	FM	FM
209	50241	TFF	13:08	13:47	FM	KH
209	50241	TFF	13:48	14:32	KH	FS
209	00000	SPD	14:32	14:36	FS	FS
209	50146	TFF	14:46	15:32	FS	KH
209	50146	PIF	15:32	15:33	KH	KH
209	00000	BEV	15:33	15:37	KH	KH
209	00000	PAU	15:42	16:12	KH	KH
209	00000	BEV	16:12	16:16	KH	KH
209	60148	PIT	16:16	16:17	KH	KH
209	60148	TFF	16:17	16:45	KH	HO
209	00000	RNG	16:45	16:47	HO	HO
209	00000	SPD	16:47	16:51	HO	HO
209	00000	RNG	16:51	16:53	HO	HO
209	60252	TFF	16:54	17:23	HO	KH
209	60252	TFF	17:24	17:50	KH	HT
209	20155	TFF	18:00	18:26	HT	KH
209	20155	PIF	18:26	18:27	KH	KH
209	00000	BEV	18:27	18:31	KH	KH
209	00000	CUD	18:31	18:41	KH	KH

Table C.2: Train driver duty representation, type II.

209	1	TogNr	55238
209	1	OpgaveType	DRIVING.TRAIN
209	1	StartTid	772
209	1	SlutTid	822
209	1	AfgStation	1
209	1	AnkStation	14
209	1	PDSRessKode	6
209	1	PDSKomm	NULL
209	1	StogAttention	NULL
209	2	TogNr	55144
209	2	OpgaveType	DRIVING.TRAIN
209	2	StartTid	837
209	2	SlutTid	888
209	2	AfgStation	14
209	2	AnkStation	1
209	2	PDSRessKode	6
209	2	PDSKomm	NULL
209	2	StogAttention	NULL
209	3	TogNr	55144
209	3	OpgaveType	DRIVING.TRAIN
209	3	StartTid	889
209	3	SlutTid	922
209	3	AfgStation	1
209	3	AnkStation	2
209	3	PDSRessKode	6
209	3	PDSKomm	NULL
209	3	StogAttention	NULL
209	4	TogNr	55248
209	4	OpgaveType	DRIVING.TRAIN
209	4	StartTid	938
209	4	SlutTid	971
209	4	AfgStation	2
209	4	AnkStation	1
209	4	PDSRessKode	13
209	4	PDSKomm	NULL
209	4	StogAttention	NULL
209	5	TogNr	NULL
209	5	OpgaveType	PAU.BREAK
209	5	StartTid	982
209	5	SlutTid	1012
209	5	AfgStation	1
209	5	AnkStation	1
209	5	PDSRessKode	NULL
209	5	PDSKomm	:AUTOMATIC
209	5	StogAttention	NULL

209	6	TogNr	60150
209	6	OpgaveType	DRIVING.TRAIN
209	6	StartTid	1017
209	6	SlutTid	1045
209	6	AfgStation	1
209	6	AnkStation	7
209	6	PDSRessKode	13
209	6	PDSKomm	NULL
209	6	StogAttention	NULL
209	7	TogNr	60254
209	7	OpgaveType	DRIVING.TRAIN
209	7	StartTid	1053
209	7	SlutTid	1083
209	7	AfgStation	7
209	7	AnkStation	1
209	7	PDSRessKode	13
209	7	PDSKomm	NULL
209	7	StogAttention	NULL
209	8	TogNr	60254
209	8	OpgaveType	DRIVING.TRAIN
209	8	StartTid	1084
209	8	SlutTid	1110
209	8	AfgStation	1
209	8	AnkStation	8
209	8	PDSRessKode	13
209	8	PDSKomm	NULL
209	8	StogAttention	NULL
209	9	TogNr	20157
209	9	OpgaveType	DRIVING.TRAIN
209	9	StartTid	1120
209	9	SlutTid	1146
209	9	AfgStation	8
209	9	AnkStation	1
209	9	PDSRessKode	3
209	9	PDSKomm	NULL
209	9	StogAttention	NULL
209	0	PDSRessKode1	3
209	0	PDSRessKode2	19
209	0	Cin	20
209	0	Cud	15
209	0	StartTid	752
209	0	SlutTid	1161
209	0	PDSIntTjNr	169117
209	0	TjenesteNr	209
209	0	Schedule	D-KH-12345
209	0	Attention	NULL
209	0	OpgaveType	

Appendix D

Station Names and Codes

Table D.1: Stations on the S-train network which appear on the passenger timetable.

1	Albertslund	ALB	44	KB Hallen	KBN
2	Allerød	LI	45	Kildebakke	KET
3	Avedøre	AVØ	46	Kildedal	KID
4	Bagsværd	BAV	47	Klampenborg	KL
5	Ballerup	BA	48	København H	KH
6	Bernstorffsvej	BFT	49	Køge	KJ
7	Birkerød	BI	50	Langgade	VAT
8	Bispebjerg	BIT	51	Lyngby	LY
9	Brøndby Strand	BSA	52	Malmparken	MPT
10	Brøndbyøster	BØT	53	Måløv	MW
11	Buddinge	BUD	54	Nordhavn	NHT
12	Charlottenlund	CH	55	Ny Ellebjerg	NEL
13	Danshøj	DAH	56	Nørrebro	NØ
14	Dybbølsbro	DBT	57	Nørreport	KN
15	Dyssegård	DYT	58	Ordrup	OP
16	Ellebjerg	ELB	59	Peter Bangs Vej	PBT
17	Emdrup	EMT	60	Ryparken	RYT
18	Enghave	AV	61	Rødovre	RDO
19	Farum	FM	62	Sjælør	SJÆ

20	Flintholm	FL	63	Skovbrynet	SKT
21	Frederikssund	FS	64	Skovlunde	SKO
22	Friheden	FRH	65	Solrød Strand	SOL
23	Fuglebakken	FUT	66	Sorgenfri	SFT
24	Gentofte	GJ	67	Stengården	SGT
25	Gl. Toftegård	GTG	68	Stenløse	ST
26	Glostrup	GL	69	Svanemøllen	SAM
27	Greve	GRE	70	Sydhavn	SYV
28	Grøndal	GHT	71	Taastrup	TÅ
29	Hareskov	HAR	72	Valby	VAL
30	Hellerup	HL	73	Vallensbæk	VLB
31	Herlev	HER	74	Vangede	ANG
32	Hillerød	HI	75	Vanløse	VAN
33	Holte	HOT	76	Veksø	VS
34	Hundige	UND	77	Vesterport	VPT
35	Husum	HUT	78	Vigerslev All	VGT
36	Hvidovre	HIT	79	Virum	VIR
37	Høje Taastrup	HTÅ	80	Værløse	VÆR
38	Ishøj	IH	81	Ølby	ØLB
39	Islev	IST	82	Ølstykke	ØL
40	Jersie	JSI	83	Østerport	KK
41	Jyllingevej	JYT	84	Ålholm	ÅLM
42	Jægersborg	JÆT	85	Åmarken	ÅM
43	Karlslunde	KLU			

Appendix E

Test Results for Initial Set of Columns Generation

Following notation is used in the tables:

- **#N** is the number of nodes in the branch-and-price tree,
- **#VarIn** is the number of columns in the initial set generated with the tested value of η_{part},
- **Time** is the computational time in seconds for solving the problem instance to optimality with the tested value of η_{part},
- $\leq \eta_z$ is **T** if the computational time for solving the problem instance with the tested value of η_{part} is less than or equal to that of $\eta_{\text{part}} = z$, and **F** otherwise,
- $\leq$all is **T** if the computational time for solving the problem instance with the tested value of η_{part} is less than or equal to that of the remaining values of η_{part}, and **F** otherwise.

Table E.1: Testing the value of $\eta_{\text{part}} = 1$.

ID	#N	#VarIn	Time	$\leq \eta_5$	$\leq \eta_{10}$	$\leq \eta_{15}$	$\leq \eta_{20}$	$\leq$all
S11	1	16	0.38	T	F	F	F	F
S12	1	25	0.39	T	T	T	T	T
S13	1	69	0.41	F	T	T	T	F
S14	1	163	0.48	F	T	T	T	F
S15	1	402	0.58	T	T	T	T	T
S16	1	1105	0.78	T	T	T	T	T
S21	1	29	0.55	F	F	F	F	F
S22	1	65	0.59	F	F	F	F	F
S23	1	120	0.47	T	F	T	T	F
S24	1	364	0.58	T	T	T	T	T
S25	1	394	0.58	T	T	T	T	T
S26	1	882	0.78	T	T	T	T	T
S31	1	30	0.81	F	F	F	F	F
S32	1	150	0.81	T	F	F	T	F
S33	1	1185	1.27	F	T	T	T	F
S34	1	3551	1.39	F	F	T	T	F
S35	1	22765	4.91	F	F	F	F	F
S36	1	118872	22.53	F	F	F	F	F
S41	1	58	1.15	T	F	F	T	F
S42	1	365	1.20	T	T	T	T	T
S43	1	1211	1.7	T	T	T	T	T
S44	1	5009	4.33	F	F	F	T	F
S45	3	46189	14.38	F	F	F	F	F
S46	1	196324	29.73	F	F	F	F	F
S51	1	46	0.86	T	T	T	F	F
S52	1	306	1.16	T	F	T	T	F
S53	1	1222	1.97	T	T	T	T	T
S54	1	4201	2.62	F	F	F	F	F
S55	1	42769	6.55	F	F	F	F	F
S56	1	212278	35.8	F	F	F	F	F
S61	1	42	1.34	F	F	F	F	F
S62	1	424	1.72	T	T	T	T	T
S63	1	1692	3.23	T	T	F	T	F
S64	1	8140	2.75	T	F	T	T	F
S65	1	35320	6.94	F	F	F	F	F
S66	1	254413	32.42	F	F	F	F	F
S71	1	85	1.66	T	T	T	T	T
S72	1	432	2.31	F	F	F	T	F
S73	1	1925	3.34	T	T	F	T	F
S74	1	8101	3.93	F	F	F	F	F
S75	1	30828	6.64	F	F	F	F	F
S76	3	271099	77.28	F	F	F	F	F
$\sum$T				20	17	19	24	11
$\sum$F				22	25	23	18	31

Table E.2: Testing the value of $\eta_{\text{part}} = 5$.

ID	#N	#VarIn	Time	$\leq \eta_1$	$\leq \eta_{10}$	$\leq \eta_{15}$	$\leq \eta_{20}$	$\leq$all
S11	1	11	0.38	T	F	F	F	F
S12	1	14	0.42	F	F	F	F	F
S13	1	22	0.39	T	T	T	T	T
S14	1	62	0.47	T	T	T	T	T
S15	1	114	0.59	F	T	T	T	F
S16	1	222	0.78	T	T	T	T	T
S21	1	15	0.45	T	F	F	F	F
S22	1	27	0.52	T	F	F	F	F
S23	1	46	0.48	F	F	T	T	F
S24	1	139	0.58	T	T	T	T	T
S25	1	140	0.67	F	T	T	T	F
S26	1	275	0.89	F	T	T	T	F
S31	1	15	0.77	T	F	F	F	F
S32	1	58	0.83	F	F	F	T	F
S33	1	266	1.23	T	T	T	T	T
S34	1	830	1.34	T	T	T	T	T
S35	1	3564	4.30	T	F	F	F	F
S36	7	12257	20.36	T	F	T	F	F
S41	1	25	1.17	F	F	F	T	F
S42	1	106	1.31	F	T	T	T	F
S43	1	281	1.95	F	F	F	T	F
S44	1	1207	4.31	T	T	F	T	F
S45	1	5996	5.50	T	T	F	T	F
S46	1	17614	14.47	T	F	T	F	F
S51	1	22	0.86	T	T	T	F	F
S52	1	88	1.25	F	F	F	F	F
S53	1	338	2.19	F	T	T	F	F
S54	1	1137	2.11	T	T	T	T	T
S55	1	5190	4.23	T	T	F	T	F
S56	1	18874	15.27	T	F	F	F	F
S61	1	26	1.30	T	F	F	F	F
S62	1	139	1.75	F	T	F	T	F
S63	1	536	3.31	F	T	F	T	F
S64	1	1828	3.17	F	F	F	F	F
S65	1	6070	5.80	T	T	F	T	F
S66	1	20452	12.70	T	F	F	T	F
S71	1	36	1.72	F	T	T	T	F
S72	1	136	2.20	T	T	T	T	T
S73	1	638	3.66	F	F	F	F	F
S74	1	1786	3.23	T	F	F	F	F
S75	1	5911	6.27	T	F	T	F	F
S76	1	26385	16.94	T	F	F	F	F
$\sum$T				26	21	19	24	8
$\sum$F				16	21	23	18	34

Table E.3: Testing the value of $\eta_{\text{part}} = 10$.

ID	#N	#VarIn	Time	$\leq \eta_1$	$\leq \eta_5$	$\leq \eta_{15}$	$\leq \eta_{20}$	$\leq$ all
S11	1	11	0.36	T	T	F	F	F
S12	1	11	0.39	T	T	T	T	T
S13	1	12	0.41	T	F	T	T	F
S14	1	28	0.61	F	F	F	F	F
S15	1	51	0.75	F	F	F	F	F
S16	1	91	0.97	F	F	F	T	F
S21	1	15	0.42	T	T	T	T	T
S22	1	20	0.48	T	T	T	T	T
S23	1	25	0.42	T	T	T	T	T
S24	1	66	0.69	F	F	F	T	F
S25	1	69	0.75	F	F	F	F	F
S26	1	125	1.11	F	F	F	T	F
S31	1	15	0.72	T	T	F	T	F
S32	1	37	0.75	T	T	T	T	T
S33	1	122	1.28	F	F	T	T	F
S34	1	409	1.38	T	F	T	T	F
S35	1	1539	4.11	T	T	F	F	F
S36	3	4359	12.72	T	T	T	T	T
S41	1	21	1.11	T	T	T	T	T
S42	1	57	1.34	F	F	T	T	F
S43	1	144	1.89	F	T	F	T	F
S44	1	533	4.31	T	T	F	T	F
S45	1	2334	5.77	T	F	F	T	F
S46	1	6353	14.11	T	T	T	F	F
S51	1	18	0.86	T	T	T	F	F
S52	1	49	1.11	T	T	T	T	T
S53	3	160	2.53	F	F	F	F	F
S54	1	509	2.11	T	T	T	T	T
S55	1	1963	4.44	T	F	F	T	F
S56	1	6285	11.69	T	T	T	T	T
S61	1	23	1.17	T	T	F	T	F
S62	1	81	1.83	F	F	F	F	F
S63	1	226	3.44	F	F	F	F	F
S64	1	709	2.59	T	T	T	T	T
S65	1	2221	6.16	T	F	F	F	F
S66	1	6482	12.17	T	T	F	T	F
S71	1	29	1.75	F	F	T	F	F
S72	1	72	2.2	T	T	T	T	T
S73	1	294	3.52	F	T	F	T	F
S74	1	781	2.94	T	T	F	T	F
S75	1	2078	5.63	T	T	T	T	T
S76	1	9186	15.42	T	T	F	F	F
$\sum$T				28	25	20	29	13
$\sum$F				14	17	22	13	29

Table E.4: Testing the value of $\eta_{\text{part}} = 15$.

ID	#N	#VarIn	Time	$\leq \eta_1$	$\leq \eta_5$	$\leq \eta_{10}$	$\leq \eta_{20}$	$\leq$all
S11	1	11	0.33	T	T	T	T	T
S12	1	11	0.41	F	T	F	T	F
S13	1	11	0.41	T	F	T	T	F
S14	1	21	0.56	F	F	T	F	F
S15	1	37	0.70	F	F	T	F	F
S16	1	53	0.94	F	F	T	T	F
S21	1	15	0.44	T	T	F	F	F
S22	1	15	0.50	T	T	F	T	F
S23	1	15	0.53	F	F	F	T	F
S24	1	40	0.67	F	F	T	T	F
S25	1	45	0.67	F	T	T	T	F
S26	1	72	0.91	F	F	T	T	F
S31	1	15	0.70	T	T	T	T	T
S32	1	25	0.80	T	T	F	T	F
S33	1	89	1.33	F	F	F	T	F
S34	1	256	1.42	F	F	F	T	F
S35	1	916	3.75	T	T	T	F	F
S36	13	2388	29.22	F	F	F	F	F
S41	1	19	1.11	T	T	T	T	T
S42	1	44	1.50	F	F	F	F	F
S43	1	100	1.86	F	T	T	T	F
S44	1	335	4.13	T	T	T	T	T
S45	1	1186	5.38	T	T	T	T	T
S46	1	3339	15.31	T	F	F	F	F
S51	1	18	0.86	T	T	T	F	F
S52	1	42	1.16	T	T	F	T	F
S53	3	103	2.44	F	F	T	F	F
S54	1	284	2.11	T	T	T	T	T
S55	1	967	4.15	T	T	T	T	T
S56	1	3138	12.44	T	T	F	T	F
S61	1	22	1.16	T	T	T	T	T
S62	1	54	1.73	F	T	T	T	F
S63	1	132	3.20	T	T	T	T	T
S64	1	388	2.75	T	T	F	T	F
S65	1	1040	5.16	T	T	T	T	T
S66	1	3278	11.63	T	T	T	T	T
S71	1	27	1.75	F	F	T	F	F
S72	1	52	2.28	T	F	F	T	F
S73	1	181	3.27	T	T	T	T	T
S74	1	461	2.92	T	T	T	T	T
S75	1	1014	6.27	T	T	F	F	F
S76	1	4690	12.72	T	T	T	T	T
$\sum$T				26	27	27	31	14
$\sum$F				16	15	15	11	28

Table E.5: Testing the value of $\eta_{\text{part}} = 20$.

ID	#N	#VarIn	Time	$\leq \eta_1$	$\leq \eta_5$	$\leq \eta_{10}$	$\leq \eta_{15}$	$\leq$ all
S11	1	11	0.34	T	T	T	F	F
S12	1	11	0.41	F	T	F	T	F
S13	1	11	0.44	F	F	F	F	F
S14	1	14	0.50	F	F	T	T	F
S15	1	26	0.61	F	F	T	T	F
S16	1	41	0.97	F	F	T	F	F
S21	1	15	0.42	T	T	T	T	T
S22	1	15	0.50	T	T	F	T	F
S23	1	15	0.55	F	F	F	F	F
S24	1	25	0.78	F	F	F	F	F
S25	1	26	0.70	F	F	T	F	F
S26	1	50	1.14	F	F	F	F	F
S31	1	15	0.72	T	T	T	F	F
S32	1	25	0.83	F	T	F	F	F
S33	1	63	1.34	F	F	F	F	F
S34	1	196	1.53	F	F	F	F	F
S35	1	528	3.72	T	T	T	T	T
S36	3	1562	29.22	T	T	F	T	F
S41	1	19	1.11	F	F	F	F	F
S42	1	33	1.50	F	F	F	T	F
S43	1	77	1.86	F	T	F	F	F
S44	1	229	4.13	F	F	F	F	F
S45	3	772	5.38	T	F	F	F	F
S46	1	2010	15.31	T	T	T	T	T
S51	1	18	0.86	T	T	T	T	T
S52	1	34	1.16	F	T	F	F	F
S53	1	84	2.44	F	T	T	T	F
S54	1	220	2.11	T	T	T	T	T
S55	1	624	4.45	T	F	F	F	F
S56	1	1966	13.72	T	T	F	F	F
S61	1	22	1.17	T	T	T	F	F
S62	1	46	1.75	F	T	T	F	F
S63	1	106	3.33	F	F	T	F	F
S64	1	272	2.75	T	T	F	T	F
S65	1	698	5.91	T	F	T	F	F
S66	1	1901	12.98	T	F	F	F	F
S71	1	27	1.73	F	F	T	T	F
S72	1	50	2.33	F	F	F	F	F
S73	1	129	3.59	F	T	F	F	F
S74	1	333	2.98	T	T	F	F	F
S75	1	663	5.67	T	T	F	T	F
S76	1	2562	13.38	T	T	T	F	F
$\sum$T				19	22	18	15	5
$\sum$F				23	20	24	27	37

Appendix F

Test Results for Pricing Strategies

Following notation is used in the tables:

- **N** is the number of nodes in the branch-and-price tree,
- **It** is the number of pricing iterations,
- **V** is the number of variables generated in the branch-and-price algorithm,
- **T** is the computational time in seconds for solving the problem instance to optimality with the tested pricing strategy,
- $\leq$**XX** is **T** if the number of iterations or the computational time for solving the problem instance with the tested pricing strategy is less than or equal to that of the pricing strategy **XX**, and **F** otherwise,
- $\leq$all is **T** if the number of iterations or the computational time for solving the problem instance with the tested pricing strategy is less than or equal to that of the other pricing strategies all together, and **F** otherwise.

Table F.1: Testing the multiple pricing strategy **MP**.

					Compare #Iterations				Compare Time			
ID	N	It	V	T	≤**SP**	≤**PP**	≤**FP**	≤all	≤**SP**	≤**PP**	≤**FP**	≤all
S11	1	10	89	0.33	T	T	T	T	T	T	T	T
S12	1	12	106	0.41	T	T	T	T	T	T	T	T
S13	1	14	124	0.41	T	T	T	T	T	T	T	T
S14	1	16	197	0.56	F	T	T	F	F	T	T	F
S15	1	20	232	0.70	T	T	T	T	T	T	T	T
S16	1	33	377	0.94	F	T	T	F	F	T	T	F
S21	1	10	111	0.44	T	T	T	T	T	T	T	T
S22	1	13	137	0.50	T	T	T	T	T	T	T	T
S23	1	17	197	0.53	T	T	T	T	T	T	T	T
S24	1	21	251	0.67	T	T	T	T	T	T	T	T
S25	1	16	240	0.67	T	T	T	T	T	T	T	T
S26	1	22	357	0.91	T	T	T	T	T	T	T	T
S31	1	18	140	0.70	T	T	T	T	T	T	T	T
S32	1	18	199	0.80	T	T	T	T	T	T	T	T
S33	1	26	423	1.33	T	T	T	T	T	T	T	T
S34	1	21	728	1.42	T	T	T	T	T	T	T	T
S35	1	28	1739	3.75	T	T	T	T	F	T	T	F
S36	13	99	3878	29.22	F	T	T	F	F	T	T	F
S41	1	24	195	1.11	T	T	T	T	T	T	T	T
S42	1	27	261	1.50	T	T	T	T	T	T	T	T
S43	1	27	460	1.86	T	T	T	T	T	T	T	T
S44	1	31	1013	4.13	T	T	T	T	T	T	T	T
S45	1	33	2164	5.38	T	T	T	T	T	T	T	T
S46	1	48	4744	15.31	F	T	T	F	F	T	T	F
S51	1	18	170	0.86	T	T	T	T	T	T	T	T
S52	1	23	258	1.16	T	T	T	T	T	T	T	T
S53	3	38	523	2.44	T	T	T	T	F	T	T	F
S54	1	21	810	2.11	T	T	T	T	T	T	T	T
S55	1	24	1685	4.15	T	T	T	T	T	T	T	T
S56	1	37	4391	12.44	T	T	T	T	F	T	T	F
S61	1	18	177	1.16	T	T	T	T	T	T	T	T
S62	1	23	365	1.73	T	T	T	T	T	T	T	T
S63	1	26	632	3.20	T	T	T	T	T	T	T	T
S64	1	24	1032	2.75	T	T	T	T	T	T	T	T
S65	1	28	1959	5.16	T	T	T	T	T	T	T	T
S66	1	34	4493	11.63	T	T	T	T	T	T	T	T
S71	1	29	278	1.75	T	T	T	T	T	T	T	T
S72	1	33	441	2.28	T	T	T	T	F	T	T	F
S73	1	26	661	3.27	T	T	T	T	T	T	T	T
S74	1	24	1053	2.92	T	T	T	T	T	T	T	T
S75	1	33	1980	6.27	T	T	T	T	F	T	T	F
S76	1	34	5934	12.72	T	T	T	T	T	T	T	T
∑T					38	42	42	38	33	42	42	33
∑F					4	0	0	4	9	0	0	9

Table F.2: Testing the limited subsequences pricing strategy **SP**.

					Compare #Iterations				Compare Time			
ID	N	It	V	T	≤**MP**	≤**PP**	≤**FP**	≤all	≤**MP**	≤**PP**	≤**FP**	≤all
S11	1	10	74	0.33	T	T	T	T	T	T	T	T
S12	1	14	86	0.42	F	T	T	F	F	T	T	F
S13	1	15	119	0.44	F	T	T	F	F	T	T	F
S14	1	15	134	0.47	T	T	T	T	T	T	T	T
S15	1	24	215	0.75	F	T	T	F	F	T	T	F
S16	1	22	264	0.80	T	T	T	T	T	T	T	T
S21	1	11	95	0.44	F	T	T	F	T	T	T	T
S22	1	17	129	0.56	F	T	T	F	F	T	T	F
S23	1	18	158	0.55	F	T	T	F	F	T	T	F
S24	1	21	222	0.73	T	T	T	T	F	T	T	F
S25	1	22	238	0.77	F	T	T	F	F	T	T	F
S26	1	29	373	1.13	F	T	T	F	F	T	T	F
S31	1	19	139	0.73	F	T	T	F	F	T	T	F
S32	1	26	214	0.95	F	T	T	F	F	T	T	F
S33	1	27	371	1.34	F	T	T	F	F	T	T	F
S34	1	32	738	1.78	F	T	T	F	F	T	T	F
S35	1	29	1673	3.20	F	T	T	F	T	T	T	T
S36	3	51	3587	11.91	T	T	T	T	T	T	T	T
S41	1	26	200	1.20	F	T	T	F	F	T	T	F
S42	1	30	254	1.52	F	T	T	F	F	T	T	F
S43	1	35	469	1.91	F	T	T	F	F	T	T	F
S44	1	41	1011	4.27	F	T	T	F	F	T	T	F
S45	3	50	2246	7.63	F	T	T	F	F	T	T	F
S46	1	40	4530	11.72	T	T	T	T	T	T	T	T
S51	1	22	170	0.92	F	T	T	F	F	T	T	F
S52	1	32	281	1.34	F	T	T	F	F	T	T	F
S53	1	39	520	2.23	F	T	T	F	T	T	T	T
S54	1	29	813	2.27	F	T	T	F	F	T	T	F
S55	1	35	1794	4.22	F	T	T	F	F	T	T	F
S56	1	39	4338	11.56	F	T	T	F	T	T	T	T
S61	1	23	182	1.30	F	T	T	F	F	T	T	F
S62	3	42	400	2.13	F	T	T	F	F	T	T	F
S63	1	41	663	3.73	F	T	T	F	F	T	T	F
S64	1	38	1078	3.61	F	T	T	F	F	T	T	F
S65	1	41	1954	5.61	F	T	T	F	F	T	T	F
S66	1	47	4492	12.45	F	T	T	F	F	T	T	F
S71	1	31	265	1.89	F	T	T	F	F	T	T	F
S72	1	37	431	2.27	F	T	T	F	T	T	T	T
S73	1	39	707	3.80	F	T	T	F	F	T	T	F
S74	1	39	1122	3.48	F	T	T	F	F	T	T	F
S75	1	37	1925	5.31	F	T	T	F	T	T	T	T
S76	1	46	6059	14.13	F	T	T	F	F	T	T	F
∑T					6	42	42	6	11	42	42	11
∑F					36	0	0	36	31	0	0	31

Table F.3: Testing the partial pricing strategy **PP**.

					Compare #Iterations				Compare Time			
ID	N	It	V	T	≤**MP**	≤**SP**	≤**FP**	≤all	≤**MP**	≤**SP**	≤**FP**	≤all
S11	1	28	67	0.56	F	F	F	F	F	F	F	F
S12	1	40	83	0.70	F	F	F	F	F	F	T	F
S13	1	56	106	1.20	F	F	F	F	F	F	T	F
S14	1	87	157	1.25	F	F	F	F	F	F	T	F
S15	1	114	207	2.42	F	F	T	F	F	F	T	F
S16	1	129	245	2.91	F	F	T	F	F	F	T	F
S21	1	36	91	0.92	F	F	T	F	F	F	T	F
S22	1	59	120	1.33	F	F	F	F	F	F	F	F
S23	1	78	146	1.14	F	F	T	F	F	F	T	F
S24	1	112	216	1.56	F	F	T	F	F	F	T	F
S25	1	98	209	1.67	F	F	T	F	F	F	T	F
S26	1	181	326	2.53	F	F	T	F	F	F	T	F
S31	1	72	137	1.67	F	F	T	F	F	F	T	F
S32	1	87	181	2.11	F	F	T	F	F	F	T	F
S33	1	149	337	2.66	F	F	T	F	F	F	T	F
S34	1	244	635	4.55	F	F	T	F	F	F	T	F
S35	1	435	1536	12.00	F	F	T	F	F	F	T	F
S36	1	555	3164	39.93	F	F	T	F	F	F	T	F
S41	3	122	190	3.20	F	F	F	F	F	F	F	F
S42	1	112	238	3.14	F	F	F	F	F	F	F	F
S43	1	178	391	3.66	F	F	T	F	F	F	T	F
S44	1	333	830	8.73	F	F	T	F	F	F	T	F
S45	1	490	1880	16.16	F	F	T	F	F	F	T	F
S46	1	636	4222	46.42	F	F	T	F	F	F	T	F
S51	1	84	160	1.70	F	F	F	F	F	F	T	F
S52	1	106	230	2.48	F	F	T	F	F	F	F	F
S53	1	223	442	4.44	F	F	T	F	F	F	T	F
S54	1	249	685	7.95	F	F	T	F	F	F	T	F
S55	1	408	1562	12.77	F	F	T	F	F	F	T	F
S56	1	625	4001	53.67	F	F	T	F	F	F	T	F
S61	1	101	198	2.77	F	F	F	F	F	F	F	F
S62	1	173	335	3.50	F	F	F	F	F	F	T	F
S63	1	250	525	6.19	F	F	T	F	F	F	T	F
S64	1	332	895	10.91	F	F	T	F	F	F	T	F
S65	1	436	1688	15.27	F	F	T	F	F	F	T	F
S66	1	646	4174	36.59	F	F	T	F	F	F	T	F
S71	1	121	239	3.70	F	F	F	F	F	F	T	F
S72	1	197	366	4.14	F	F	T	F	F	F	T	F
S73	1	235	562	6.31	F	F	T	F	F	F	T	F
S74	1	285	917	11.17	F	F	T	F	F	F	T	F
S75	1	495	1718	23.16	F	F	T	F	F	F	T	F
S76	1	663	5600	44.81	F	F	T	F	F	F	T	F
∑T					0	0	31	0	0	0	36	0
∑F					42	42	11	42	42	42	6	42

Table F.4: Testing the full pricing strategy **FP**.

					Compare #Iterations				Compare Time			
ID	N	It	V	T	≤**MP**	≤**SP**	≤**PP**	≤all	≤**MP**	≤**SP**	≤**PP**	≤all
S11	1	24	63	0.52	F	F	T	F	F	F	T	F
S12	1	39	82	0.75	F	F	T	F	F	F	F	F
S13	1	55	105	1.20	F	F	T	F	F	F	T	F
S14	1	78	148	1.75	F	F	T	F	F	F	F	F
S15	1	114	207	2.61	F	F	T	F	F	F	F	F
S16	1	179	295	4.36	F	F	F	F	F	F	F	F
S21	1	41	96	1.31	F	F	F	F	F	F	F	F
S22	1	54	115	1.28	F	F	T	F	F	F	T	F
S23	1	81	149	1.20	F	F	F	F	F	F	F	F
S24	1	134	238	2.20	F	F	F	F	F	F	F	F
S25	1	111	222	1.98	F	F	F	F	F	F	F	F
S26	1	216	361	5.38	F	F	F	F	F	F	F	F
S31	1	73	138	1.73	F	F	F	F	F	F	F	F
S32	1	93	187	2.27	F	F	F	F	F	F	F	F
S33	1	188	376	4.52	F	F	F	F	F	F	F	F
S34	1	352	743	13.14	F	F	F	F	F	F	F	F
S35	1	700	1801	43.55	F	F	F	F	F	F	F	F
S36	3	1138	3713	133.47	F	F	F	F	F	F	F	F
S41	1	102	188	2.67	F	F	T	F	F	F	T	F
S42	1	98	224	2.27	F	F	T	F	F	F	T	F
S43	1	215	428	5.88	F	F	F	F	F	F	F	F
S44	1	475	972	18.17	F	F	F	F	F	F	F	F
S45	1	857	2247	61.67	F	F	F	F	F	F	F	F
S46	1	1186	4772	175.30	F	F	F	F	F	F	F	F
S51	1	79	155	2.14	F	F	T	F	F	F	F	F
S52	1	124	248	2.36	F	F	F	F	F	F	T	F
S53	1	257	476	7.11	F	F	F	F	F	F	F	F
S54	1	373	809	11.50	F	F	F	F	F	F	F	F
S55	1	548	1702	36.66	F	F	F	F	F	F	F	F
S56	1	1017	4393	164.69	F	F	F	F	F	F	F	F
S61	1	82	179	2.53	F	F	T	F	F	F	T	F
S62	1	151	313	3.55	F	F	T	F	F	F	F	F
S63	1	327	602	9.61	F	F	F	F	F	F	F	F
S64	1	390	953	14.56	F	F	F	F	F	F	F	F
S65	1	690	1942	42.78	F	F	F	F	F	F	F	F
S66	1	998	4526	139.59	F	F	F	F	F	F	F	F
S71	1	117	235	3.75	F	F	T	F	F	F	F	F
S72	1	265	434	5.69	F	F	F	F	F	F	F	F
S73	1	286	613	10.50	F	F	F	F	F	F	F	F
S74	1	393	1025	18.81	F	F	F	F	F	F	F	F
S75	1	694	1917	43.48	F	F	F	F	F	F	F	F
S76	1	1243	6180	156.58	F	F	F	F	F	F	F	F
∑T					0	0	12	0	0	0	7	0
∑F					42	42	30	42	42	42	35	42

Bibliography

E. Abbink, M. Fischetti, L. Kroon, G. Timmer, and M. Vromans. Reinventing crew scheduling at Netherlands Railways. *Interfaces*, 35(5):393–401, 2005.

E. Abbink, J. van 't Wout, and D. Huisman. Solving large scale crew scheduling problems by using iterative partitioning. In *ATMOS 2007, 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization and Systems*, pages 96–106, 2007.

E. Abbink, J. van 't Wout, and D. Huisman. Solving large scale crew scheduling problems by using iterative partitioning. Econometric Institute Report EI2008-03, Econometric Institute, Erasmus University Rotterdam, The Netherlands, 2008.

A. Abdelghany, G. Ekollu, R. Narasimhan, and K. Abdelghany. A proactive crew recovery decision support tool for commercial airlines during irregular operations. *Annals of Operations Research*, 127:309–331, 2004.

B. Adenso-Díaz, M. O. González, and P. González-Torre. On-line timetable re-scheduling in regional train services. *Transportation Research Part B*, 33(6):387–398, 1999.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, Inc., 1993.

A. C. Andersen, C. G. Christensen, I. L. T. Heilmann, and C. T. Holst.

Planlægning af køreplaner. Bachelor thesis, Informatics & Mathematical Modelling, Technical University of Denmark, 2006.

Y. Aneja, V. Aggarwal, and K. Nair. Shortest chain subject to side constraints. *Networks*, 13(2):295–302, 1983.

A. Assad. Analytical models in rail transportation: an annotated bibliography. *INFOR. Canadian Journal of Operational Research and Information Processing*, 19(1):59–80, 1981.

A. A. Assad. Models for rail transportation. *Transportation Research Part A*, 14(3):205–220, 1980.

F. Azevedo, P. Barahona, F. Fages, and F. Rossi, editors. *Recent Advances in Constraints: 11th Annual ERCIM International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2006 Revised Selected and Invited Papers*, volume 4651 LNAI, 2007. Springer Verlag.

M. Ball, L. Bodin, and R. Dall. A matching based heuristic for scheduling mass transit crews and vehicles. *Transportation Science*, 17(1):4–31, 1983.

C. Barnhart, N. L. Boland, L. W. Clarke, E. L. Johnson, and G. L. Nemhauser. Flight string models for aircraft fleeting and routing. *Transportation Science*, 32(3):208–220, 1998a.

C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998b.

C. Barnhart, A. M. Cohn, E. L. Johnson, D. Klabjan, G. L. Nemhauser, and P. H. Vance. Airline crew scheduling. In R. W. Hall, editor, *Handbook of Transportation Science*. Kluwer Academic Publishers, second edition, 2003.

L. Bengtsson, R. Galia, T. Gustafsson, C. Hjorring, and N. Kohl. Railway crew pairing optimization. Technical Report CRTR-0408, Carmen Research and Technology Report, Nov. 2004.

C. Berge. Balanced matrices. *Mathematical Programming*, 2(1):19–31, 1972.

N. L. Boland, L. W. Clarke, and G. L. Nemhauser. The asymmetric traveling salesman problem with replenishment arcs. *European Journal of Operational Research*, 123:408–427, 2000.

G. Budai, G. Maróti, R. Dekker, D. Huisman, and L. Kroon. Re-scheduling in railways: the rolling stock balancing problem. Econometric Institute Report EI2007-21, Econometric Institute, Erasmus University Rotterdam, The Netherlands, May 2007.

M. R. Bussieck, T. Winter, and U. T. Zimmermann. Discrete optimization in public rail transport. *Mathematical Programming*, 79:415–444, 1997.

E. R. Butchers, P. R. Day, A. P. Goldie, S. Miller, J. A. Meyer, D. M. Ryan, A. C. Scott, and C. A. Wallace. Optimized crew scheduling at Air New Zealand. *Interfaces*, 31(1):30–56, 2001.

X. Cai, C. Goh, and A. I. Mees. Greedy heuristics for rapid scheduling of trains on a single track. *IIE Transactions*, 30(5):481, 1998. Institute of Industrial Engineers.

G. Caimi, D. Burkolter, and T. Herrmann. Finding delay–tolerant train routings through stations. In H. Fleuren, editor, *Operations Research Proceedings 2004: Selected Papers of the Annual International Conference of the German Operations Research Society (GOR)*, pages 136–143. Springer Verlag, 2005.

A. Caprara, M. Fischetti, P. Toth, D. Vigo, and P. L. Guida. Algorithms for railway crew management. *Mathematical Programming*, 79:125–141, 1997.

A. Caprara, F. Focacci, E. Lamma, P. Mello, M. Milano, P. Toth, and D. Vigo. Integrating Constraint Logic Programming and Operations Research techniques for the crew rostering problem. *Software - Practice and Experience*, 28(1):49–76, 1998a.

A. Caprara, P. Toth, D. Vigo, and M. Fischetti. Modeling and solving the crew rostering problem. *Operations Research*, 46(6):820–830, 1998b.

A. Caprara, M. Fischetti, P. L. Guida, P. Toth, and D. Vigo. Solution of large-scale railway crew planning problem: the Italian experience. In M. H. M.

Wilson, editor, *Computer-Aided Transit Scheduling*, volume 471 of *Lecture Notes in Economic and Mathematical Systems*, pages 1–18. Springer-Verlag, 1999a.

A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Operations Research*, 47(5):730–743, 1999b.

A. Caprara, M. Monaci, and P. Toth. A global method for crew planning in railway applications. In S. Voß and J. D. Daduna, editors, *Computer-Aided Scheduling of Public Transport*, volume 505 of *Lecture Notes in Economic and Mathematical Systems*, pages 17–36. Springer-Verlag, Berlin, 2001.

A. Caprara, L. Kroon, M. Monaci, M. Peeters, and P. Toth. Passenger railway optimization. In C. Barnhart and G. Laporte, editors, *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*, pages 129–187. Elsevier, 2007.

A. J. Castro and E. Oliveira. Using specialized agents in a distributed mas to solve airline operations problems: A case study. In *2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'07)*, pages 473–476, 2007a.

A. J. Castro and E. Oliveira. A distributed multi-agent system to solve airline operations problems. In *ICEIS 2007. Proceedings of the Ninth International Conference on Enterprise Information Systems*, volume AIDSS, pages 22–30, Funchal, Madeira, Portugal, June 12 - 16 2007b.

S. C. K. Chu and E. C. H. Chan. Crew scheduling of light rail transit in Hong Kong: from modeling to implementation. *Computers & Operations Research*, 25(11):887–894, 1998.

V. Chvátal. *Linear programming*. W. H. Freeman and Company, New York, 1983.

J. Clausen, A. Larsen, J. Larsen, and N. Rezanova. Disruption management in the airline industry–concepts, models and methods. *Computers & Operations Research*, in press, 2009. doi: 10.1016/j.cor.2009.03.027.

M. Conforti, G. Cornuéjols, A. Kapoor, and K. V. sković. Perfect, ideal and balanced matrices. *European Journal of Operational Research*, 133(3): 455–461, 2001.

J.-F. Cordeau, P. Toth, and D. Vigo. A survey of optimization models for train routing and scheduling. *Transportation Science*, 32(4):380–404, 1998.

G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

A. D'Ariano. *Improving Real-Time Train Dispatching: Models, Algorithms and Applications*. PhD thesis, Department of Transport & Planning, Delft University of Technology, April 2008.

A. D'Ariano, D. Pacciarelli, and M. Pranzo. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, 183(2):643–657, 2007.

J.-L. Deng. Control problems of grey systems. *Systems & Control Letters*, 1 (5):288–294, 1982. ISSN 01676911.

G. Desaulniers, J. Desrosiers, Y. Dumas, S. Marc, B. Rioux, M. M. Solomon, and F. Soumis. Crew pairing at Air France. *European Journal of Operational Research*, 97:245–259, 1997.

M. Desrochers and F. Soumis. A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR*, 26(3):191–212, 1988.

J. Desrosiers, Y. Dumas, M. Solomon, and F. Soumis. Time constrained routing and scheduling. In M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, editors, *Handbooks in Operations Research and Management Science*, volume 8: Network Routing, pages 35–140. Elsevier, Amsterdam, 1995.

I. Dumitrescu and N. Boland. Algorithms for the weight constrained shortest path problem. *International Transactions in Operational Research*, 8(1): 15–29, 2001.

A. T. Ernst, M. Krishnamoorthy, and D. Dowling. Train crew rostering using simulated annealing. In *Proceedings of International Conference on Optimization Techniques and Applications, ICOTA'98*, pages 859–866, Perth, 1998.

A. T. Ernst, H. Jiang, M. Krishnamoorthy, H. Nott, and D. Sier. An optimization approach to train crew rostering. In *Proceedings of the 15th National Conference of the Australian Society for Operations Research*, pages 470–481, Gold Cost, 1999.

A. T. Ernst, H. Jiang, M. Krishnamoorthy, H. Nott, and D. Sier. Rail crew scheduling and rostering optimization algorithms. In S. Voß and J. D. Daduna, editors, *Computer-Aided Scheduling of Public Transport*, volume 505 of *Lecture Notes in Economic and Mathematical Systems*, pages 53–71. Springer-Verlag, Berlin, 2001a.

A. T. Ernst, H. Jiang, M. Krishnamoorthy, H. Nott, and D. Sier. An integrated optimization model for train crew management. *Annals of Operations Research*, 108:211–224, 2001b.

A. T. Ernst, H. Jiang, M. Krishnamoorthy, B. Owens, and D. Sier. An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, 127:21–144, 2004.

P. J. Fioole, L. Kroon, G. Maróti, and A. Schrijver. A rolling stock circulation model for combining and splitting of passenger trains. *European Journal of Operational Research*, 174:1281–1297, 2006.

M. Fischetti and L. G. Kroon. Solving large-scale crew scheduling problems at the Dutch Railways. In S. Voß and J. D. Daduna, editors, *Computer-Aided Scheduling of Public Transport*, volume 505 of *Lecture Notes in Economic and Mathematical Systems*. Springer-Verlag, Berlin, 2001.

M. Folkmann, J. Jespersen, and M. N. Nielsen. Estimates on rolling stock and crew in DSB S-tog based on timetables. In F. G. et al., editor, *Algorithmic Methods for Railway Optimization*, number 4359 in Lecture Notes in Computer Science. Springer-Verlag, 2007. Paper draft.

P. Føns. Decision support for depot planning in the railway industry. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2006.

S. Fores. *Column generation approaches to bus driver scheduling*. PhD thesis, University of Leeds, 1996.

S. Fores, L. G. Proll, and A. Wren. An improved ILP system for driver scheduling. In M. H. M. Wilson, editor, *Computer-Aided Transit Scheduling*, volume 471 of *Lecture Notes in Economic and Mathematical Systems*, pages 43–62. Springer-Verlag, 1999.

S. Fores, L. Proll, and A. Wren. Experiences with a flexible driver scheduler. In S. Voß and J. D. Daduna, editors, *Computer-Aided Scheduling of Public Transport*, volume 505 of *Lecture Notes in Economic and Mathematical Systems*, pages 137–152. Springer-Verlag, Berlin, 2001.

S. Fores, L. Proll, and A. Wren. TRACS II: a hybrid IP/heuristic driver scheduling system for public transport. *Journal of the Operational Research Society*, 53(10):1093–1100, 2002.

R. Freling, R. M. Lentink, and M. A. Odijk. Scheduling train crews: a case study for the Dutch Railways. In S. Voß and J. D. Daduna, editors, *Computer-Aided Scheduling of Public Transport*, volume 505 of *Lecture Notes in Economic and Mathematical Systems*, pages 153–165. Springer-Verlag, Berlin, 2001.

R. Freling, R. M. Lentink, and A. P. M. Wagelmans. A decision support system for crew planning in passenger transportation using a flexible branch-and-price algorithm. *Annals of Operations Research*, 127:203–222, 2004.

M. Gamache, F. Soumis, G. Marquis, and J. Desrosiers. A column generation approach for large-scale aircrew rostering problems. *Operations Research*, 47(2):247–263, 1999.

A. Ginkel and A. Schbel. To wait or not to wait? the bicriteria delay management problem in public transportation. *Transportation Science*, 41(4): 527, 2007.

J. Goossens. *Models and Algorithms for Railway Line Planning Problems*. PhD thesis, University of Maastricht, The Netherlands, 2004.

Y. Guo. A decision support framework for the airline crew schedule disruption management with strategy mapping. In *Operations Research Proceedings 2004*, volume Volume 2004. Springer Berlin Heidelberg, 2005a.

Y. Guo. *Decision Support Systems for Airline Crew Recovery*. PhD thesis, University of Paderborn, April 2005b.

A. E. Haghani. Rail freight transportation: a review of recent optimization models for train routing and empty car distribution. *Journal of Advanced Transportation*, 21(2):147–172, 1987.

A. Hartog, D. Huisman, E. J. W. Abbink, and L. G. Kroon. Decision support for crew rostering at NS. *Public Transport: Planning and Operations*, accepted for publication, 2008.

G. Heilporn, L. De Giovanni, and M. Labbe. Optimization models for the single delay management problem in public transportation. *European Journal of Operational Research*, 189(3):762–774, 2008.

T. M. Herrman. *Stability of Timetables and Train Routings the Station Regions*. PhD thesis, Swiss Federal Insitute of Technology Zurich, 2006.

F. S. Hillier and G. J. Lieberman. *Introduction to Operations Research*. McGraw-Hill, 1995.

C. Hjorring. Solving larger crew pairing problems. In *Proceedings of TRISTAN V: The Fifth Triennial Symposium on Transportation Analysis, Le Gosier, Guadeloupe*, Handbooks in Operations Research and Management Science, June 2004.

C. A. Hjorring and J. Hansen. Column generation with a rule modeling language for airline crew pairing. In *Proceedings of 34th Annual Conference of the Operations Research Society of New Zealand*, Hamilton, New Zealand, December 1999.

A. J. Hoffman and J. B. Kruskal. Integral boundary points of convex polyhedra. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*, number 38 in Annals of Mathematics Studies, pages 223–246. Princeton University Press, Princeton, 1956.

M. Hofman and L. F. Madsen. Robustness in train scheduling. Master thesis, Informatics & Mathematical Modelling, Technical University of Denmark, 2005.

M. A. Hofman, L. Madsen, J. J. Groth, J. Clausen, and J. Larsen. Robustness and recovery in train scheduling - a simulation study from DSB S-tog A/S. Technical Report IMM-Technical Report-2006-12, Informatics & Mathematical Modelling, Technical University of Denmark, 2006.

J. Hu and E. L. Johnson. Computational results with a primal-dual subproblem simplex method. *Operations Research Letters*, 25:149–157, 1999.

D. Huisman. A column generation approach for the rail crew re-scheduling problem. *European Journal of Operational Research*, 180(1):163–173, 2007.

D. Huisman, L. G. Kroon, R. M. Lentink, and M. J. C. M. Vromans. Operations Research in passenger railway transportation. *Statistica Neerlandica*, 59(4):467–497, 2005.

J. Jacobs. Reducing delays by means of computer-aided "on the spot" rescheduling. In J. Allan, C. Brebbia, R. Hill, G. Sciutto, and S. Sone, editors, *Computers in Railways IX*, pages 603–612. WIT Press, 2004.

J. Jespersen Groth. Simulation study of train driver schedule. In *Nordic Optimization Symposium, Copenhagen*, April 2006.

J. Jespersen Groth. *Decision Support for the Rolling Stock Dispatcher*. PhD thesis, Technical University of Denmark, 2008a.

J. Jespersen Groth. The rolling stock recovery problem. Technical Report IMM-Technical Report-2008-18, Informatics & Mathematical Modelling, Technical University of Denmark, 2008b.

J. Jespersen Groth, J. Clausen, and J. Larsen. Optimal reinsertion of cancelled train lines. Technical Report IMM-Technical Report-2006-13, Informatics & Mathematical Modelling, Technical University of Denmark, 2006.

J. Jespersen Groth, D. Potthoff, J. Clausen, D. Huisman, L. Kroon, G. Maróti, and M. N. Nielsen. Disruption management in passenger railway transportation. Econometric Institute Report EI2007-05, Econometric Institute, Erasmus University Rotterdam, The Netherlands, 2007.

E. L. Johnson, L. Lettovsky, G. L. Nemhauser, R. Pandit, and S. Querido. Final report to Northwest Airlines on the crew recovery problem. Technical report, The Logistic Institute, Georgia Institute of Technology, Atlanta, GA, August 12 1994.

N. Kohl. Solving the world's largest crew scheduling problem. *ORbit, Newsletter Danish Operations Research Society*, Special Issue:8–12, August 2003.

N. Kohl and S. E. Karisch. Airline crew rostering: problem types, modeling, and optimization. *Annals of Operations Research*, 127:223–257, 2004.

L. Kroon and M. Fischetti. Crew scheduling for Netherlands Railways "Destination: Customer". In S. Voß and J. D. Daduna, editors, *Computer-Aided Scheduling of Public Transport*, volume 505 of *Lecture Notes in Economic and Mathematical Systems*, pages 181–201. Springer-Verlag, Berlin, 2001.

L. Kroon and M. Fischetti. Scheduling train drivers and guards: the Dutch "Noord-Oost" Case. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, 2000.

L. Kroon, D. Huisman, E. Abbink, P. J. Fioole, M. Fischetti, G. Maróti, L. Shrijver, A. Steenbeek, and R. Ybema. The new dutch timetable: The or revolution. *Interfaces*, 39:6–17, 2009.

L. G. Kroon, R. Dekker, and M. J. C. M. Vromans. Cyclic railway timetabling: a stochastic optimization approach. In *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

A. S. Kwan, R. S. Kwan, M. E. Parker, and A. Wren. Producing train driver schedules under different operating strategies. In N. H. M. Wilson, editor, *Computer-Aided Transit Scheduling*, volume 471 of *Lecture Notes in Economic and Mathematical Systems*, pages 129–154. Springer-Verlag, 1999a.

A. S. K. Kwan, R. S. K. Kwan, M. E. Parker, and A. Wren. Producing train driver shifts by computer. In J. Allan, C. Brebbia, R. Hill, G. Sciutto, and S. Sone, editors, *Computers in Railways V - Vol.1 Railway Systems and Management*, pages 421–435. Computational Mechanics Publications, 1996.

A. S. K. Kwan, R. S. Kwan, and A. Wren. Driver scheduling using genetic algorithms with embedded combinatorial traits. In M. H. M. Wilson, editor, *Computer-Aided Transit Scheduling*, volume 471 of *Lecture Notes in Economic and Mathematical Systems*, pages 81–102. Springer-Verlag, 1999b.

A. S. K. Kwan, M. E. Parker, R. S. K. Kwan, S. Fores, L. Proll, and A. Wren. Recend advances in TRACS. In *9th International Conference*

on Computer-Aided Scheduling of Public Transport (CASPT), San Diego, California, Aug. 2004.

R. S. K. Kwan, A. Wren, and A. S. K. Kwan. Hybrid genetic algorithms for scheduling bus and train drivers. In *Proceedings of the 2000 Congress on Evolutonary Computation*, number 1 in Evolutionary Computation, pages 285–292, 2000.

R. S. K. Kwan, A. S. R. Kwan, and A. Wren. Evolutionary driver scheduling with relief chains. *Evolutionary Computation*, 9(4):445–460, 2001.

C.-K. Lee. The integrated scheduling and rostering problem of train driver using Genetic algorithm. In *9th International Conference on Computer-Aided Scheduling of Public Transport (CASPT), San Diego, California*, Aug. 2004.

C.-K. Lee and C.-H. Chen. Scheduling of train driver for taiwan railway administration. *Journal of the Eastern Asia Society for Transportation Science*, 5:292–306, 2003.

A. Lehman. On the width-length inequality. *Mathematical Programming*, 17: 403–417, 1979.

R. M. Lentink, M. A. Odijk, and E. van Rijn. Crew rostering for the High Speed Train. Erim report series research in management, Erasmus Research Institute of Management, Erasmus University Rotterdam, The Netherlands, 2002.

L. Lettovsky, E. L. Johnson, and G. L. Nemhauser. Airline crew recovery. *Transportation Science*, 34(4):337–348, 2000.

C. Liebchen. *Periodic Timetable Optimization in Public Transport*. PhD thesis, Technische Universitt Berlin, 2006.

C. Liebchen. The first optimized railway timetable in practice. *Transportation Science*, 42(4):420–435, 2008.

C. Liebchen and S.Stiller. Delay resistant timetabling. Technical Report ARRIVAL-TR-0056, ARRIVAL Project, November 2006. Presented at CASPT 2006.

L. Lovász. Normal hypergraphs and the perfect graph conjecture. *Discrete Mathematics*, 306(10-11):867–875, 2006.

L. Lovász. Normal hypergraphs and the perfect graph conjecture. *Discrete Mathematics*, 2:253–268, 1972.

M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

R. Lusby. *Optimization methods for routing trains through railway junctions*. PhD thesis, Department of Engineering Science, School of Engineering, The University of Auckland, New Zealand, 2008.

G. Maróti. *Operations Research Models for Railway Rolling Stock Planning*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2006.

J. Martins, E. Morgado, and R. Haugen. Tpo: a system for scheduling and managing train crew in norway. In J. Riedl and R. Hill, editors, *Proceedings of the Fifteenth Innovative Applications of Artificial Intelligence Conference*, pages 25–32. AAAI Press, 2003.

C. P. Medard and N. Sawhney. Airline crew scheduling from planning to operations. Technical Report CRTR-0406, Carmen Research and Technology Report, June 2004.

C. P. Medard and N. Sawhney. Airline crew scheduling from planning to operations. *European Journal of Operational Research*, 183(3):1013–1027, 2007.

M. Meilton. Selecting and implementing a computer aided scheduling system for a large bus company. In S. Voß and J. D. Daduna, editors, *Computer-Aided Scheduling of Public Transport*, volume 505 of *Lecture Notes in Economic and Mathematical Systems*, pages 203–214. Springer-Verlag, Berlin, 2001.

A. Mercier and F. Soumis. An integrated aircraft routing, crew scheduling and flight retiming model. *Computers & Operations Research*, 34:2251 – 2265, 2007.

E. M. Morgado and J. P. Martins. Scheduling and managing crew in the portuguese railways. *Expert Systems with Applications*, 5:301–321, 1992.

E. M. Morgado and J. P. Martins. An AI-based approach to crew scheduling. In *Proceedings of 9th IEEE Conference on Artificial Intelligence for Applications*, pages 71–77, 1993.

E. M. Morgado and J. P. Martins. Crews-NS: scheduling train crews in the Netherlands. *AI Magazine*, 19(1):25–38, 1998a.

E. M. Morgado and J. P. Martins. Crews: a train scheduling tool. In *Proceedings of the International Conference on Computer Aided Design, Manufacture and Operation in The Railway and Other Advanced Mass Transit Systems*, pages 287–297. Computational Mechanics Publ, 1998b.

MOSEK. *The Mosek .NET API Manual. Version 5.0, www.mosek.com.* The Mosek Optimization Software, Denmark, 2008.

G. L. Nemhauser, M. W. Savelsbergh, and G. C. Sigismondi. MINTO, a Mixed INTeger optimizer. *Operations Research Letters*, 15(1):47–58, 1994.

L. K. Nielsen. A decision support framework for rolling stock rescheduling. Technical Report ARRIVAL-TR-0158, Arrival Project, August 2008.

L. K. Nielsen. Planlægning af arbejdsplaner for togrevisorer i S-toge. Master's thesis, Department of Mathematics and Computer Science, University of Southern Denmark, 2006.

M. Nielsen, B. Hove, and J. Clausen. Constructing periodic timetables using MIP - a case study from DSB S-train. *European Journal of Operational Research*, 1(3):213–227, 2006.

M. N. Nielsen and T. Christensen. Duty planning and design of experiments. In *ATMOS 2006, 6th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization and Systems*, 2006.

R. Nissen and K. Haase. Duty-period-based network model for crew rescheduling in European airlines. *Journal of Scheduling*, 9(3):255–278, 2006.

W. Orchard-Hays. *Advanced Linear–Programming Computing Techniques.* McGraw-Hill, New York, 1968.

M. Padberg. Lehman's forbidden minor characterization of ideal 0-1 matrices. *Discrete Mathematics*, 111:409–420, 1993.

M. W. Padberg. Perfect zero-one matrices. *Mathematical Programming*, 6 (2):180–196, 1974.

M. E. Parker, A. Wren, and R. S. K. Kwan. Modelling the scheduling of train drivers. In J. D. Daduna, I. Branco, and J. M. P. Paixao, editors, *Computer-Aided Transit Scheduling*, volume 430 of *Lecture Notes in Economic and Mathematical Systems*, pages 359–370. Springer-Verlag, 1995.

L. W. P. Peeters. *Cyclic railway timetable optimization*. PhD thesis, Erasmus Research Institute of Management, Erasmus University Rotterdam, The Netherlands, 2003.

L. Ping, N. Axin, J. Limin, and W. Fuzhang. Study on intelligent train dispatching. In *Proceedings of 2001 IEEE Intelligent Transportation Systems*, pages 949–953, 2001.

D. Potthoff, D. Huisman, and G. Desaulniers. Column generation with dynamic duty selection for railway crew rescheduling. Technical Report EI 2008-28, Econometric Institute, Erasmus University Rotterdam, The Netherlands, December 19, 2008.

N. J. Rezanova and D. M. Ryan. The train driver recovery problem – a set partitioning based model and solution method. Technical Report IMM-Technical Report-2006-24, Informatics & Mathematical Modelling, Technical University of Denmark, 2006.

N. J. Rezanova and D. M. Ryan. The train driver recovery problem – a set partitioning based model and solution method. *Computers & Operations Research*, in press, 2009. doi: 10.1016/j.cor.2009.03.023.

J. Rodriguez. A constraint programming model for real-time train scheduling at junctions. *Transportation Research Part B*, 41(2):231–245, 2007.

D. M. Ryan. ZIP - a zero-one integer programming package for scheduling. Computer Science and Systems Division Report 85, Atomic Energy Research Establishment, Harwell, Oxfordshire, UK, 1980.

D. M. Ryan. The solution of massive generalized set partitioning problems in aircrew rostering. *Journal of the Operational Research Society*, 43(5): 459–467, 1992.

D. M. Ryan and J. C. Falkner. On the integer properties of scheduling set partitioning models. *European Journal of Operational Research*, 35(3): 442–456, 1988.

D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Aided Scheduling of Public Transport*, pages 269–280. North-Holland Publishing Company, 1981.

T. L. Saaty. *The Analytic Hierarchy Process*. McGraw-Hill, 1980.

I. Şahin. Railway traffic control and train scheduling based on inter-train conflict management. *Transportation Research Part B*, 33(7):511–534, 1999.

A. Schbel. Integer programming approaches for solving the delay management problem. In *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

B. M. Smith and A. Wren. A bus crew scheduling system using a set covering formulation. *Transportation Research Part A*, 22(2):97–108, 1988.

B. M. Smith, C. J. Layfield, A. Wren, E. C. Freuder, and R. J. Wallace. A constraint programming pre-processor for a bus driver scheduling system. In *Constraint Programming and Large Scale Discrete Optimization. DIMACS Workshop*, pages 131–148, 2001.

M. S. Sodhi and S. Norris. A flexible, fast, and optimal modeling approach applied to crew rostering at London Underground. *Annals of Operations Research*, 127:259–281, 2004.

M. Song, G. Wei, and G. Yu. A Decision Support Framework for Crew Management During Airline Irregular Operations. In Y. G, editor, *Operations Research in the Airline Industry*. Kluwer Academic Publishers, Boston, 1998.

M. Stojković and F. Soumis. An optimization model for the simultaneous operational flight and pilot scheduling problem. *Management Science*, 47 (9):1290 – 1305, 2001.

M. Stojković and F. Soumis. The operational flight and multi-crew scheduling problem. *Yugoslavian Journal of Operations Research*, 15(1):25 – 48, 2005.

M. Stojković, F. Soumis, and J. Desrosiers. The operational airline crew scheduling problem. *Transportation Science*, 32(3):232–245, 1998.

B. Szpigel. Optimal train scheduling on a single line railway. *Operations Research*, 72:344–351, 1973.

D. Teodorović and G. Stojković. Model to reduce airline schedule disturbances. *Journal of Transportation Engineering*, 121:324–331, 1995.

J. Törnquist. Railway traffic disturbance management - an experimental analysis of distrurbance complexity, management objectives and limitations in planning horizon. *Transportation Research Part A*, 41(3):249–266, 2007.

J. Törnquist and J. A. Persson. Train traffic deviation handling using tabu search and simulated annealing. In *Proceedings of the 38th Hawaii International Conference on System Sciences*, 2005.

J. Törnquist and J. A. Persson. N-tracked railway traffic re-scheduling during disturbances. *Transportation Research Part B*, 41(3):342–362, 2007.

B. Vaidyanathan, K. C. Jha, and R. K. Ahuja. Multicommodity network flow approach to the railroad crew-scheduling problem. *IBM Journal of Research and Development*, 51(3/4):325–344, 2007.

J. C. Villumsen. Construction of timetables based on periodic event scheduling. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2006.

M. Vromans. *Reliability of Railway Systems*. PhD thesis, Erasmus University Rotterdam, Rotterdam School of Management, The Netherlands, 2005.

C. G. Walker, J. N. Snowdon, and D. M. Ryan. Simultaneous disruption recovery of a trian timetable and crew roster in real time. *Computers & Operations Research*, 32(8):2077–2094, 2005.

D. Wedelin. An algorithm for large scale 0-1 integer programming with application to airline crew scheduling. *Annals of Operations Research*, 57 (1):283–301, 1995.

S. Wegele and E. Schnieder. Dispatching of train operations using genetic algorithms. In *CD-ROM Proceedings of the 1st International Seminar on Railway Operations Modelling and Analysis*, Delft, The Netherlands, 2005.

G. Wei, G. Yu, and M. Song. Optimization model and algorithm for crew management during airline irregular opertions. *Journal of Combinatorial Optimization*, 1:305–321, 1997.

O. Weide, D. Ryan, and M. Ehrgott. An iterative approach to robust and integrated aircraft routing and crew scheduling. *Computers & Operations Research*, in press, 2009. doi:10.1016/j.cor.2009.03.024.

W. E. Wilhelm. A technical review of column generation in integer programming. *Optimization and Engineering*, 2:159–200, 2001.

W. P. Willers. *Improved algorithms for bus crew scheduling.* PhD thesis, University of Leeds, 1995.

W. P. Willers, L. Proll, and A. Wren. A dual strategy for solving the linear programming relaxation of a driver scheduling system. *Annals of Operations Research*, 58:519–531, 1995.

N. Wilson, editor. *Computer-Aided Transit Scheduling*, volume 471 of *Lecture Notes in Economic and Mathematical Systems.* Springer-Verlag, Berlin, 1999.

L. A. Wolsey. *Integer programming.* John Wiley & Sons, Inc., 1998.

A. Wren. Scheduling vehicles and their drivers - forty years' experience. In *9th International Conference on Computer-Aided Scheduling of Public Transport (CASPT), San Diego, California*, Aug. 2004.

A. Wren and R. S. K. Kwan. Installing an urban transport scheduling system. *Journal of Scheduling*, 2(1):3–17, 1999.

A. Wren and B. M. Smith. Experiences with a crew scheduling system based on set covering. In J. D. Daduna and A. Wren, editors, *Computer-Aided Transit Scheduling*, pages 104–118. Springer-Verlag, 1988.

A. Wren, R. S. K. Kwan, and M. E. Parker. Scheduling of rail driver duties. In T. K. S. Murthy, B. Mellit, C. Brebbia, G. Sciutto, and S. Sone, editors,

Railway Operations, volume 2 of *Computers in Railways - IV*. Computational Mechanics Publications, Southampton, Boston, 1994.

A. Wren, S. Fores, A. S. K. Kwan, R. S. K. Kwan, M. Parker, and L. Proll. A flexible system for scheduling drivers. *Journal of Scheduling*, 6(5):437–455, 2003.

G. Yu, M. Argüello, G. Song, S. M. McCowan, and A. White. A new era for crew recovery at Continental Airlines. *Interfaces*, 33(1):5–22, 2003.

X. L. Zhao, J. F. Zhu, and M. Guo. Application of grey programming in irregular flights scheduling. In *2007 IEEE International Conference on Industrial Engineering and Engineering Management*, 2007.